

tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer[†]

Released 2024-02-29

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	10
7	Label and Reference commands	11
8	Setup label attributes	11
9	Commands to fill seq and prop	12
10	General tagging commands	12
11	Keys for tagpdfsetup	14
12	loading of engine/more dependent code	15
I	The tagpdf-checks module	
	Messages and check code	
	Part of the tagpdf package	17
1	Commands	17

*This file describes v0.98x, last revised 2024-02-29.

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

2	Description of log messages	17
2.1	\ShowTagging command	17
2.2	Messages in checks and commands	18
2.3	Messages from the ptagging code	18
2.4	Warning messages from the lua-code	18
2.5	Info messages from the lua-code	18
2.6	Debug mode messages and code	19
2.7	Messages	19
3	Messages	21
3.1	Messages related to mc-chunks	21
3.2	Messages related to structures	22
3.3	Attributes	23
3.4	Roles	23
3.5	Miscellaneous	24
4	Retrieving data	25
5	User conditionals	25
6	Internal checks	26
6.1	checks for active tagging	26
6.2	Checks related to structures	26
6.3	Checks related to roles	28
6.4	Check related to mc-chunks	28
6.5	Checks related to the state of MC on a page or in a split stream	31
II The tagpdf-user module		
Code related to L^AT_EX₂ε user commands and document commands		
Part of the tagpdf package		35
1	Setup commands	35
2	Commands related to mc-chunks	35
3	Commands related to structures	36
4	Debugging	36
5	Extension commands	37
5.1	Fake space	37
5.2	Tagging of paragraphs	37
5.3	Header and footer	38
5.4	Link tagging	38
6	Socket support	38
7	User commands and extensions of document commands	39
8	Setup and preamble commands	39

9	Commands for the mc-chunks	39
10	Commands for the structure	40
11	Socket support	41
12	Debugging	41
13	Commands to extend document commands	45
	13.1 Document structure	45
	13.2 Structure destinations	46
	13.3 Fake space	46
	13.4 Paratagging	46
	13.5 Header and footer	54
	13.6 Links	56

III The tagpdf-tree module

Commands trees and main dictionaries

	Part of the tagpdf package	58
--	-----------------------------------	-----------

1	Trees, pdfmanagement and finalization code	58
	1.1 Check structure	58
	1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction	59
	1.3 Writing the IDtree	60
	1.4 Writing structure elements	61
	1.5 ParentTree	62
	1.6 Rolemap dictionary	65
	1.7 Classmap dictionary	65
	1.8 Namespaces	66
	1.9 Finishing the structure	67
	1.10 StructParents entry for Page	67

IV The tagpdf-mc-shared module

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

	Part of the tagpdf package	68
--	-----------------------------------	-----------

1	Public Commands	68
2	Public keys	69
3	Marked content code – shared	70
	3.1 Variables and counters	70
	3.2 Functions	71
	3.3 Keys	74

V	The tagpdf-mc-generic module	
	Code related to Marked Content (mc-chunks), generic mode	
	Part of the tagpdf package	75
1	Marked content code – generic mode	75
1.1	Variables	75
1.2	Functions	76
1.3	Looking at MC marks in boxes	79
1.4	Keys	87
VI	The tagpdf-mc-luacode module	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the tagpdf package	89
1	Marked content code – luamode code	89
1.1	Commands	91
1.2	Key definitions	95
VII	The tagpdf-struct module	
	Commands to create the structure	
	Part of the tagpdf package	98
1	Public Commands	98
2	Public keys	99
2.1	Keys for the structure commands	99
2.2	Setup keys	101
3	Variables	101
3.1	Variables used by the keys	103
3.2	Variables used by tagging code of basic elements	104
4	Commands	104
4.1	Initialization of the StructTreeRoot	105
4.2	Adding the /ID key	106
4.3	Filling in the tag info	107
4.4	Handlings kids	108
4.5	Output of the object	111
5	Keys	115
6	User commands	121
7	Attributes and attribute classes	129
7.1	Variables	129
7.2	Commands and keys	129

VIII	The tagpdf-luatex.def	
	Driver for luatex	
	Part of the tagpdf package	133
1	Loading the lua	133
2	Logging functions	137
3	Helper functions	139
	3.1 Retrieve data functions	139
	3.2 Functions to insert the pdf literals	141
4	Function for the real space chars	143
5	Function for the tagging	147
6	Parenttree	152
IX	The tagpdf-roles module	
	Tags, roles and namespace code	
	Part of the tagpdf package	154
1	Code related to roles and structure names	154
	1.1 Variables	155
	1.2 Namespaces	157
	1.3 Adding a new tag	158
	1.3.1 pdf 1.7 and earlier	159
	1.3.2 The pdf 2.0 version	161
	1.4 Helper command to read the data from files	163
	1.5 Reading the default data	165
	1.6 Parent-child rules	165
	1.6.1 Reading in the csv-files	166
	1.6.2 Retrieving the parent-child rule	168
	1.7 Remapping of tags	173
	1.8 Key-val user interface	173
X	The tagpdf-space module	
	Code related to real space chars	
	Part of the tagpdf package	175
1	Code for interword spaces	175
	Index	178

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

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

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

`activate/spaces_<key>`

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

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

Keys to activate the various tagging steps.

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

The key allows to suppress the creation of structure destinations

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

`activate/tagunmarked_<key>`
`tagunmarked_<key>` (deprecated)

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

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

`tagstruct`
`tagstructobj`
`tagabspage`
`tagmcabs`
`tagmcid`

These are attributes used by the label/ref system.

1 Initialization and test if pdfmanagement is active.

```
1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2024-02-29} {0.98x}
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} {2024-02-29} {0.98x}
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>
```

Debug mode has its special mapping:

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

```
39 <*base>
40 \ProvidesExplPackage {tagpdf-base} {2024-02-29} {0.98x}
41 {part of tagpdf - provide base, no-op versions of the user commands }
42 </base>
```

3 Package options

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

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

4 Packages

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

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

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

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

4.1 a LastPage label

See also issue #2 in Accessible-xref

`__tag_lastpagelabel:`

```
69 (*package)
70 \cs_new_protected:Npn \__tag_lastpagelabel:
71 {
72   \legacy_if:nT { @filesw }
73   {
```

```

74     \exp_args:NNne \exp_args:NNe\iow_now:Nn \@auxout
75     {
76         \token_to_str:N \new@label@record
77         {@tag@LastPage}
78         {
79             {abspage} { \int_use:N \g_shipout_readonly_int}
80             {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
81             {tagstruct}{\int_use:N \c@g__tag_struct_abs_int }
82         }
83     }
84 }
85 }
86
87 \AddToHook{enddocument/afterlastpage}
88   {\__tag_lastpagelabel:}

```

(End of definition for __tag_lastpagelabel:.)

5 Variables

A few temporary variables

```

\l__tag_tmpa_tl
\l__tag_tmpb_tl
\l__tag_get_tmpc_tl
\tag_get_parent_tmpb_tl\l__tag_tmpa_str
\l__tag_tmpa_prop
\l__tag_tmpa_seq
\l__tag_tmpb_seq
\l__tag_tmpa_clist
\l__tag_tmpa_int
\l__tag_tmpa_box
\l__tag_tmpb_box
89 \tl_new:N \l__tag_tmpa_tl
90 \tl_new:N \l__tag_tmpb_tl
91 \tl_new:N \l__tag_get_tmpc_tl
92 \tl_new:N \l__tag_get_parent_tmpa_tl
93 \tl_new:N \l__tag_get_parent_tmpb_tl
94 \str_new:N \l__tag_tmpa_str
95 \prop_new:N \l__tag_tmpa_prop
96 \seq_new:N \l__tag_tmpa_seq
97 \seq_new:N \l__tag_tmpb_seq
98 \clist_new:N \l__tag_tmpa_clist
99 \int_new:N \l__tag_tmpa_int
100 \box_new:N \l__tag_tmpa_box
101 \box_new:N \l__tag_tmpb_box

```

(End of definition for \l__tag_tmpa_tl and others.)

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

```

\c__tag_property_mc_clist
\c__tag_property_struct_clist
102 \clist_const:Nn \c__tag_property_mc_clist {tagabspage,tagmcabs,tagmcid}
103 \clist_const:Nn \c__tag_property_struct_clist {tagstruct,tagstructobj}

```

(End of definition for \c__tag_property_mc_clist and \c__tag_property_struct_clist.)

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

```

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

```

105 \bool_new:N \g__tag_active_space_bool
106 \bool_new:N \g__tag_active_mc_bool
107 \bool_new:N \g__tag_active_tree_bool
108 \bool_new:N \g__tag_active_struct_bool
109 \bool_new:N \g__tag_active_struct_dest_bool
110 \bool_gset_true:N \g__tag_active_struct_dest_bool

```

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

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

```

111 \bool_new:N \l__tag_active_mc_bool
112 \bool_set_true:N \l__tag_active_mc_bool
113 \bool_new:N \l__tag_active_struct_bool
114 \bool_set_true:N \l__tag_active_struct_bool
115 \bool_new:N \l__tag_active_socket_bool

```

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

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

```

116 \bool_new:N \g__tag_tagunmarked_bool

```

(End of definition for `\g__tag_tagunmarked_bool`.)

6 Variants of l3 commands

```

117 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F,TF}
118 \cs_generate_variant:Nn \pdf_object_ref:n {e}
119 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nne}
120 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nee,oe}
121 \cs_generate_variant:Nn \prop_gput:Nnn {Nee,Nen} %** unneeded
122 \cs_generate_variant:Nn \prop_put:Nnn {Nee} %** unneeded
123 \cs_generate_variant:Nn \prop_item:Nn {No,Ne} %** unneeded
124 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne} %** unneeded
125 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
126 \cs_generate_variant:Nn \clist_map_inline:nn {on}

```

7 Label and Reference commands

To ease transition to properties we setup internal definition. They can be replaced by the property definitions once that is released. ****** do it!

```

\__tag_property_new:nnnn At first a command to define new properties
\__tag_property_gset:nnnn 127 \cs_new_eq:NN \__tag_property_new:nnnn \property_new:nnnn
\__tag_property_ref:nnn For the non-shipout code we need also the option to reset property
128 \cs_new_eq:NN \__tag_property_gset:nnnn \property_gset:nnnn
The command to reference while giving a local default.
129 \cs_new_eq:NN \__tag_property_ref:nnn \property_ref:nnn
130 \cs_new_eq:NN \__tag_property_ref:nn \property_ref:nn
The command to record
131 \cs_new_protected:Npn \__tag_property_record:nn #1#2
132 {
133 \@bsphack
134 \property_record:nn{#1}{#2}
135 \@esphack
136 }
137

```

And a few variants

```

138 \cs_generate_variant:Nn \__tag_property_ref:nnn {enn}
139 \cs_generate_variant:Nn \__tag_property_ref:nn {en}
140 \cs_generate_variant:Nn \__tag_property_record:nn {en,eV}

```

(End of definition for __tag_property_new:nnnn, __tag_property_gset:nnnn, and __tag_property_ref:nnn.)

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

```

141 \cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
142 {
143 \__tag_property_ref:nnn {@tag@LastPage}{#1}{#2}
144 }

```

(End of definition for __tag_property_ref_lastpage:nn.)

8 Setup label attributes

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

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

```

145 \__tag_property_new:nnnn
146 { tagstruct } { now }
147 {0} { \int_use:N \c@g__tag_struct_abs_int }
148 \__tag_property_new:nnnn { tagstructobj } { now } {}
149 {

```

```

150     \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
151     {
152         \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
153     }
154 }
155 \__tag_property_new:nmmm
156 { tagabspage } { shipout }
157 {0} { \int_use:N \g_shipout_readonly_int }
158 \__tag_property_new:nmmm { tagmcabs } { now }
159 {0} { \int_use:N \c@g__tag_MCID_abs_int }
160
161 \flag_new:n { __tag/mcid }
162 \__tag_property_new:nmmm {tagmcid} { shipout }
163 {0} { \flag_height:n { __tag/mcid } }
164

```

(End of definition for `tagstruct` and others. These functions are documented on page 6.)

9 Commands to fill `seq` and `prop`

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

```

    \__tag_prop_new:N
\__tag_prop_new_linked:N 165 \cs_set_eq:NN \__tag_prop_new:N      \prop_new:N
    \__tag_seq_new:N      166 \cs_set_eq:NN \__tag_prop_new_linked:N \prop_new_linked:N
\__tag_prop_gput:Nnn     167 \cs_set_eq:NN \__tag_seq_new:N        \seq_new:N
\__tag_seq_gput_right:Nn 168 \cs_set_eq:NN \__tag_prop_gput:Nnn     \prop_gput:Nnn
    \__tag_seq_item:cn    169 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
\__tag_prop_item:cn     170 \cs_set_eq:NN \__tag_seq_item:cn       \seq_item:cn
    \__tag_seq_show:N     171 \cs_set_eq:NN \__tag_prop_item:cn      \prop_item:cn
\__tag_prop_show:N      172 \cs_set_eq:NN \__tag_seq_show:N       \seq_show:N
    \__tag_prop_show:N    173 \cs_set_eq:NN \__tag_prop_show:N      \prop_show:N
174 % cnx temporary needed for latex-lab-graphic code
175 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nen , Nee, Nne , cnn, cen, cne, cno, cnx}
176 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Ne , No, cn, ce }
177 \cs_generate_variant:Nn \__tag_prop_new:N { c }
178 \cs_generate_variant:Nn \__tag_seq_new:N { c }
179 \cs_generate_variant:Nn \__tag_seq_show:N { c }
180 \cs_generate_variant:Nn \__tag_prop_show:N { c }
181 \end{package}

```

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

10 General tagging commands

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

When stop/start pairs are nested we do not want the inner start command to restart tagging. To control this we use a local int: The stop command will increase it. The starting will decrease it and only restart tagging, if it is zero. This will replace the label version.

```

\l__tag_tag_stop_int 182 <*package | debug>
183 <package>\int_new:N \l__tag_tag_stop_int

184 \cs_set_protected:Npn \tag_stop:
185 {
186 <debug> \msg_note:nxx {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }
187 \int_incr:N \l__tag_tag_stop_int
188 \bool_set_false:N \l__tag_active_struct_bool
189 \bool_set_false:N \l__tag_active_mc_bool
190 \bool_set_false:N \l__tag_active_socket_bool
191 \__tag_stop_para_ints:
192 }
193 \cs_set_protected:Npn \tag_start:
194 {
195 \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
196 \int_if_zero:nT { \l__tag_tag_stop_int }
197 {
198 \bool_set_true:N \l__tag_active_struct_bool
199 \bool_set_true:N \l__tag_active_mc_bool
200 \bool_set_true:N \l__tag_active_socket_bool
201 \__tag_start_para_ints:
202 }
203 <debug> \msg_note:nxx {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }
204 }
205 \cs_set_eq:NN\tagstop\tag_stop:
206 \cs_set_eq:NN\tagstart\tag_start:
207 \cs_set_protected:Npn \tag_stop:n #1
208 {
209 <debug> \msg_note:nxxx {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }{#1}
210 \int_incr:N \l__tag_tag_stop_int
211 \bool_set_false:N \l__tag_active_struct_bool
212 \bool_set_false:N \l__tag_active_mc_bool
213 \bool_set_false:N \l__tag_active_socket_bool
214 \__tag_stop_para_ints:
215 }
216 \cs_set_protected:Npn \tag_start:n #1
217 {
218 \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
219 \int_if_zero:nT { \l__tag_tag_stop_int }
220 {
221 \bool_set_true:N \l__tag_active_struct_bool
222 \bool_set_true:N \l__tag_active_mc_bool
223 \bool_set_true:N \l__tag_active_socket_bool
224 \__tag_start_para_ints:
225 }
226 <debug> \msg_note:nxxx {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }{#1}
227 }
228 </package | debug>
229 (*base)
230 \cs_new_protected:Npn \tag_stop: {}

```

```

231 \cs_new_protected:Npn \tag_start:{}
232 \cs_new_protected:Npn \tagstop{}
233 \cs_new_protected:Npn \tagstart{}
234 \cs_new_protected:Npn \tag_stop:n #1 {}
235 \cs_new_protected:Npn \tag_start:n #1 {}
236 </base>

```

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

11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

`activate/mc□(setup-key)` Keys to (globally) activate tagging. `activate/spaces` activates the additional parsing needed for interword spaces. It is defined in `tagpdf-space`. `activate/struct-dest` allows

`activate/tree□(setup-key)` to activate or suppress structure destinations.

`activate/struct□(setup-key)`

`activate/all□(setup-key)`

`activate/struct-dest□(setup-key)`

```

237 (*package)
238 \keys_define:nm { __tag / setup }
239 {
240   activate/mc      .bool_gset:N = \g__tag_active_mc_bool,
241   activate/tree    .bool_gset:N = \g__tag_active_tree_bool,
242   activate/struct .bool_gset:N = \g__tag_active_struct_bool,
243   activate/all     .meta:n =
244     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
245   activate/all     .default:n = true,
246   activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

old, deprecated names

```

247   activate-mc      .bool_gset:N = \g__tag_active_mc_bool,
248   activate-tree    .bool_gset:N = \g__tag_active_tree_bool,
249   activate-struct .bool_gset:N = \g__tag_active_struct_bool,
250   activate-all     .meta:n =
251     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
252   activate-all     .default:n = true,
253   no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,

```

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

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

```

254   debug/show      .choice:,

```

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

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

`debug/uncompress□(setup-key)`

```

log□(deprecated)      255   debug/log      .choice:,
uncompress□(deprecated) 256   debug/log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
257   debug/log / v   .code:n =
258     {
259       \int_set:Nn \l__tag_loglevel_int { 1 }
260       \cs_set_protected:Nn \__tag_check_typeof_v:n { \iow_term:e {##1} }
261     },
262   debug/log / vv   .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},

```

```

263 debug/log / vvv .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
264 debug/log / all .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
265 debug/uncompress .code:n = { \pdf_uncompress: },

```

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

```

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

```

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

`activate/tagunmarked_□` (setup-key)
`tagunmarked_□` (deprecated)

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

```

268 activate/tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
269 activate/tagunmarked .initial:n = true,

```

deprecated name

```

270 tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

`page/tabsorder_□` (setup-key)
`tabsorder_□` (deprecated)

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.

```

271 page/tabsorder .choice:,
272 page/tabsorder / row .code:n =
273   \pdfmanagement_add:nnn { Page } {Tabs}{/R},
274 page/tabsorder / column .code:n =
275   \pdfmanagement_add:nnn { Page } {Tabs}{/C},
276 page/tabsorder / structure .code:n =
277   \pdfmanagement_add:nnn { Page } {Tabs}{/S},
278 page/tabsorder / none .code:n =
279   \pdfmanagement_remove:nn {Page} {Tabs},
280 page/tabsorder .initial:n = structure,

```

deprecated name

```

281 tabsorder .meta:n = {page/tabsorder={#1}},
282 }

```

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

12 loading of engine/more dependent code

```

283 \sys_if_engine luatex:T
284 {
285   \file_input:n {tagpdf-luatex.def}
286 }
287 </package>
288 < *mcloding >
289 \bool_if:NTF \g__tag_mode_lua_bool
290 {
291   \RequirePackage {tagpdf-mc-code-lua}
292 }
293 {

```

```
294 \RequirePackage {tagpdf-mc-code-generic} %
295 }
296 </mcloding>
297 <*debug>
298 \bool_if:NTF \g__tag_mode_lua_bool
299 {
300 \RequirePackage {tagpdf-debug-lua}
301 }
302 {
303 \RequirePackage {tagpdf-debug-generic} %
304 }
305 </debug>
```

Part I

The `tagpdf-checks` module

Messages and check code

Part of the `tagpdf` package

1 Commands

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

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

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

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

`\tag_if_box_tagged:NTF` * This tests if a box contains tagging commands. It relies currently on that the code, that saved the box, correctly sets the command `\l_tag_box_\int_use:N #1_t1` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

2 Description of log messages

2.1 `\ShowTagging` command

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

2.2 Messages in checks and commands

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

2.3 Messages from the ptgging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

2.7 Messages

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

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

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

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

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

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

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2024-02-29} {0.98x}
4 {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 </header>
```

3 Messages

3.1 Messages related to mc-chunks

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

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

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

mc-tag-missing If the tag is missing

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

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

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

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

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

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

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

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

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

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

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

mc-pushed Informational messages about mc-pushing.

mc-popped

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

(End of definition for mc-pushed and mc-popped. These functions are documented on page 19.)

mc-current Informational messages about current mc state.

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

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

3.2 Messages related to structures

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

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

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

struct-no-objnum Should not happen ...

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

(End of definition for struct-no-objnum. This function is documented on page 20.)

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

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

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

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

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

(End of definition for struct-faulty-nesting. This function is documented on page 20.)

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

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

(End of definition for struct-missing-tag. This function is documented on page 20.)

struct-used-twice

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

(End of definition for struct-used-twice. This function is documented on page 20.)

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

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

(End of definition for struct-label-unknown. This function is documented on page 20.)

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

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

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

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 }
49 </package>
```

(End of definition for tree-struct-still-open. This function is documented on page 20.)

The following messages are only needed in debug mode.

show-struct This two messages are used to show the current structures in the log and terminal.
show-kids

```
50 <*debug>
51 \msg_new:nnn { tag/debug } { show-struct }
52 {
53   =====\\
54   The~structure~#1~
55   \tl_if_empty:nTF {#2}
56   { is~empty \\>~ . }
57   { contains: #2 }
58   \\
59 }
60 \msg_new:nnn { tag/debug } { show-kids }
61 {
62   The~structure~has~the~following~kids:
63   \tl_if_empty:nTF {#2}
64   { \\>~ NONE }
65   { #2 }
66   \\
67   =====
68 }
69 </debug>
```

(End of definition for show-struct and show-kids. These functions are documented on page 20.)

3.3 Attributes

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

attr-unknown

```
70 <*package>
71 \msg_new:nnn { tag } {attr-unknown} { attribute~#1~is~unknown}
```

(End of definition for attr-unknown. This function is documented on page 20.)

3.4 Roles

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

```
72 \msg_new:nnn { tag } {role-missing} { tag~#1~has~no~role~assigned }
73 \msg_new:nnn { tag } {role-unknown} { role~#1~is~not~known }
74 \msg_new:nnn { tag } {role-unknown-tag} { tag~#1~is~not~known }
75 \msg_new:nnn { tag } {role-unknown-NS} { \tl_if_empty:nTF{#1}{Empty-NS}{NS~#1~is~not~known} }
```

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

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

```
76 \msg_new:nnn { tag } {role-parent-child}
77   { Parent-Child~'~#1'~-->~'~#2'.\\Relation-is~#3~\msg_line_context: }
```

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

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

```
78 \msg_new:nnn { tag } {role-remapping}
79   { remapping~tag-to~#1 }
```

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

role-tag Info messages.

```
new-tag 80 \msg_new:nnn { tag } {role-tag}           { mapping~tag~#1~to~role~#2 }
81 \msg_new:nnn { tag } {new-tag}           { adding~new~tag~#1 }
82 \msg_new:nnn { tag } {read-namespace}    { reading~namespace~definitions~tagpdf~ns~
#1.def }
83 \msg_new:nnn { tag } {namespace-missing}{ namespace~definitions~tagpdf~ns~#1.def~not~found }
84 \msg_new:nnn { tag } {namespace-unknown}{ namespace~#1~is~not~declared }
```

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

3.5 Miscellaneous

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

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

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

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

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

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

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

```
89 \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 breaks the structure.

```
90 \msg_new:nnnn { tag } {para-hook-count-wrong}
91   {The~number~of~automatic~begin~(#1)~and~end~(#2)~#3~para~hooks~differ!}
92   {This~quite~probably~a~coding~error~and~the~structure~will~be~wrong!}
93 \</package>
```

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

4 Retrieving data

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

```
94 <base>\cs_new:Npn \tag_get:n #1 { \use:c {_tag_get_data_#1: } }
```

(End of definition for \tag_get:n. This function is documented on page 17.)

5 User conditionals

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

```
95 <*base>
96 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
97   { \prg_return_false: }
98 </base>
99 <*package>
100 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF , F }
101   {
102     \bool_lazy_all:nTF
103       {
104         {\g__tag_active_struct_bool}
105         {\g__tag_active_mc_bool}
106         {\g__tag_active_tree_bool}
107         {\l__tag_active_struct_bool}
108         {\l__tag_active_mc_bool}
109       }
110     {
111       \prg_return_true:
112     }
113     {
114       \prg_return_false:
115     }
116   }
117 </package>
```

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

`\tag_if_box_tagged_p:N` This tests if a box contains tagging commands. It relies on that the code that saved the box correctly set `\l_tag_box_<box number>_t1` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.
`\tag_if_box_tagged:NTF`

```
118 <*base>
119 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
120   {
121     \tl_if_exist:cTF {l_tag_box_\int_use:N #1_t1}
122     {
123       \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_t1}}>{0}
124       { \prg_return_true: }
125       { \prg_return_false: }
126     }
127   }
```

```

127     {
128     \prg_return_false:
129     % warning??
130     }
131   }
132 </base>

```

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

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

`__tag_check_if_active_mc:TF` This checks if mc are active.

```

\__tag_check_if_active_struct:TF
133 (*package)
134 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
135   {
136     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
137     {
138       \prg_return_true:
139     }
140     {
141       \prg_return_false:
142     }
143   }
144 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
145   {
146     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
147     {
148       \prg_return_true:
149     }
150     {
151       \prg_return_false:
152     }
153   }

```

(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.

```

154 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
155   {
156     \prop_if_in:cnF { g__tag_struct_#1_prop }
157     {S}
158     {
159       \msg_error:mn { tag } {struct-missing-tag}
160     }
161   }

```

(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.

```
162 \cs_new_protected:Npn \__tag_check_structure_tag:N #1
163   {
164     \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
165     {
166       \msg_warning:nne { tag } {role-unknown-tag} {#1}
167     }
168   }
```

(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.

```
169 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
170   {
171     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
172     {
173       \msg_info:nnm { tag } {struct-show-closing} {#1}
174     }
175   }
176
177 \cs_generate_variant:Nn \__tag_check_info_closing_struct:n {o,e}
```

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

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

```
178 \cs_new_protected:Npn \__tag_check_no_open_struct:
179   {
180     \msg_error:nn { tag } {struct-faulty-nesting}
181   }
```

(End of definition for `__tag_check_no_open_struct:.`)

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

```
182 \cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
183   {
184     \prop_get:cnNT
185     {g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
186     {P}
187     \l__tag_tmpa_tl
188     {
189       \msg_warning:nnn { tag } {struct-used-twice} {#1}
190     }
191   }
```

(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.
192 \\cs_new_protected:Npn \\_tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
193 {
194   \\tl_if_empty:nTF {#2}
195   {
196     \\msg_error:nnn { tag } {role-missing} {#1}
197   }
198   {
199     \\prop_get:NnNTF \\g__tag_role_tags_NS_prop {#2} \\l_tmpa_tl
200     {
201       \\int_compare:nNnT {\\l__tag_loglevel_int} > { 0 }
202       {
203         \\msg_info:nnnn { tag } {role-tag} {#1} {#2}
204       }
205     }
206     {
207       \\msg_error:nnn { tag } {role-unknown} {#2}
208     }
209   }
210 }
```

Similar with a namespace

```
211 \\cs_new_protected:Npn \\_tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
212 {
213   \\tl_if_empty:nTF {#2}
214   {
215     \\msg_error:nnn { tag } {role-missing} {#1}
216   }
217   {
218     \\prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \\l_tmpa_tl
219     {
220       \\int_compare:nNnT {\\l__tag_loglevel_int} > { 0 }
221       {
222         \\msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
223       }
224     }
225     {
226       \\msg_error:nnn { tag } {role-unknown} {#2/#3}
227     }
228   }
229 }
```

(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).

```
230 \\cs_new_protected:Npn \\_tag_check_mc_if_nested:
231 {
232   \\_tag_mc_if_in:T
233   {
```

```

234     \msg_warning:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
235   }
236 }
237
238 \cs_new_protected:Npn \__tag_check_mc_if_open:
239 {
240   \__tag_mc_if_in:F
241   {
242     \msg_warning:nne { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
243   }
244 }

```

(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.

```

245 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
246 {
247   \int_compare:nNnT
248     { \l__tag_loglevel_int } = { 2 }
249     { \msg_info:nne {tag}{mc-#1}{#2} }
250   \int_compare:nNnT
251     { \l__tag_loglevel_int } > { 2 }
252     {
253       \msg_info:nne {tag}{mc-#1}{#2}
254       \seq_log:N \g__tag_mc_stack_seq
255     }
256 }

```

(End of definition for __tag_check_mc_pushed_popped:nn.)

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

```

257 \cs_new_protected:Npn \__tag_check_mc_tag:N #1 %#1 is var with a tag name in it
258 {
259   \tl_if_empty:NT #1
260   {
261     \msg_error:nne { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
262   }
263   \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
264   {
265     \msg_warning:nne { tag } {role-unknown-tag} {#1}
266   }
267 }

```

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

```

268 \cs_new_protected:Npn \__tag_check_init_mc_used:
269 {
270   \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
271   \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
272 }

(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.

```

273 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid abscnt
274 {
275   \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
276   {
277     \__tag_check_init_mc_used:
278     \intarray_gset:Nnn \g__tag_check_mc_used_intarray
279       {#1}
280     { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
281     \int_compare:nNnT
282       {
283         \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
284       }
285       >
286       { 1 }
287       {
288         \msg_warning:nnn { tag } {mc-used-twice} {#1}
289       }
290   }
291 }

(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.

```

292 \cs_new_protected:Npn \__tag_check_show_MCID_by_page:
293 {
294   \tl_set:Ne \l__tag_tmpa_tl
295   {
296     \__tag_property_ref_lastpage:nn
297     {abspage}
298     {-1}
299   }
300   \int_step_inline:nnnn {1}{1}
301   {
302     \l__tag_tmpa_tl
303   }
304   {
305     \seq_clear:N \l_tmpa_seq
306     \int_step_inline:nnnn
307     {1}
308     {1}
309     {
310       \__tag_property_ref_lastpage:nn
311       {tagmcabs}
312       {-1}
313     }
314   }

```

```

315         \int_compare:nT
316         {
317             \__tag_property_ref:enn
318             {mci-####1}
319             {tagabspage}
320             {-1}
321         =
322         ##1
323     }
324     {
325         \seq_gput_right:Ne \l_tmpa_seq
326         {
327             Page##1-####1-
328             \__tag_property_ref:enn
329             {mci-####1}
330             {tagmci}
331             {-1}
332         }
333     }
334 }
335 \seq_show:N \l_tmpa_seq
336 }
337 }

```

(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.

```

338 \prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
339 {
340     \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
341     { \prg_return_false: }
342     { \prg_return_true: }
343 }

```

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

```

344 \prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
345 {
346     \bool_if:nTF

```

```

347 {
348   \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
349   ||
350   \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
351 }
352 { \prg_return_true: }
353 { \prg_return_false: }
354 }

```

(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.

```

355 \prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
356 {
357   \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
358   { \prg_return_true: }
359   { \prg_return_false: }
360 }

```

(End of definition for `__tag_check_if_mc_tme_missing:TF`.)

```
361 </package>
```

```
362 <*debug>
```

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

```

363 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_info]
364 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg_line_context:] }
365
366 \cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
367 {
368   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
369   {
370     \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
371   }
372 }
373 \cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
374 {
375   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
376   {
377     \msg_note:nnnn { tag / debug } {mc-begin} {ignored} { #1 }
378   }
379 }
380 \cs_new_protected:Npn \__tag_debug_mc_end_insert:
381 {
382   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
383   {
384     \msg_note:nnn { tag / debug } {mc-end} {inserted}
385   }
386 }
387 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
388 {
389   \int_compare:nNnT { \l__tag_loglevel_int } > {0}

```

```

390     {
391     \msg_note:nnn { tag / debug } {mc-end } {ignored}
392     }
393 }

```

And now something for the structures

```

394 \msg_new:nnn { tag / debug } {struct-begin}
395 {
396   Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context:
397 }
398 \msg_new:nnn { tag / debug } {struct-end}
399 {
400   Struct~end~#1~[\msg_line_context:]
401 }
402 \msg_new:nnn { tag / debug } {struct-end-wrong}
403 {
404   Struct~end~'1'~doesn't~fit~start~'2'~[\msg_line_context:]
405 }
406 }
407 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
408 {
409   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
410   {
411     \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
412     \seq_log:N \g__tag_struct_tag_stack_seq
413   }
414 }
415 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
416 {
417   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
418   {
419     \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
420   }
421 }
422 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
423 {
424   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
425   {
426     \msg_note:nnn { tag / debug } {struct-end} {inserted}
427     \seq_log:N \g__tag_struct_tag_stack_seq
428   }
429 }
430 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
431 {
432   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
433   {
434     \msg_note:nnn { tag / debug } {struct-end } {ignored}
435   }
436 }
437 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
438 {
439   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
440   {
441     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
442     {

```

```

443     \str_if_eq:eeF
444     {#1}
445     {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
446     {
447         \msg_warning:nnee { tag/debug }{ struct-end-wrong }
448         {#1}
449         {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
450     }
451 }
452 }
453 }

```

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

```

454 \msg_new:nnn { tag / debug } {tag-stop}
455 {
456     \int_if_zero:nTF
457     {#1}
458     {Tagging~stopped}
459     {Tagging~(not)~stopped~(already~inactive)}\
460     level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
461 }
462 \msg_new:nnn { tag / debug } {tag-start}
463 {
464     \int_if_zero:nTF
465     {#1}
466     {Tagging~restarted}
467     {Tagging~(not)~restarted}\
468     level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
469 }
470 </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

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

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

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

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

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

2 Commands related to mc-chunks

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

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

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

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

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

3 Commands related to structures

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

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

4 Debugging

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

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

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

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

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

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

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

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

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

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

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

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

5 Extension commands

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

The commands and keys should be view as experimental!

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

5.1 Fake space

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

5.2 Tagging of paragraphs

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

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

The `para/tagging` key can be used in `\tagpdfsetup` and enable/disable tagging of paragraphs. `debug/show=para` puts small colored numbers at the begin and end of a paragraph. This is meant as a debugging help. The number are boxes and have a (tiny) height, so they can affect typesetting.

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

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

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

5.3 Header and footer

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

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

5.4 Link tagging

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

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

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

6 Socket support

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

The next L^AT_EX (2024-06-01) will use special sockets for the tagging.

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

Given that we sometimes have to suspend tagging, it would be fairly inefficient to put different plugs into these sockets whenever that happens. We therefore offer `\UseTaggingSocket` which is like `\UseSocket` except that the socket name is specified without `tagsupport/`, i.e.,

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

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

It is possible to use the tagging support sockets with `\UseSocket` directly, but in this case the socket remains active if e.g. `\SuspendTagging` is in force. There may be reasons for doing that but in general we expect to always use `\UseTaggingSocket`.

The L3 programming layer versions `\tag_socket_use:n` and `\tag_socket_use:nn` are slightly more efficient than `\UseTaggingSocket` because they do not have to determine how many arguments the socket takes when disabling it.

7 User commands and extensions of document commands

```

1 <@@=tag>
2 < *header>
3 \ProvidesExplPackage {tagpdf-user} {2024-02-29} {0.98x}
4   {tagpdf - user commands}
5 </header>

```

8 Setup and preamble commands

`\tagpdfsetup`

```

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

```

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

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

`\tagtool`

```

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

```

(End of definition for `\tag_tool:n` and `\tagtool`. These functions are documented on page 35.)

9 Commands for the mc-chunks

`\tagmcbegin`

`\tagmcend`

`\tagmcuse`

```

22 (*base)
23 \NewDocumentCommand \tagmcbegin { m }
24   {
25     \tag_mc_begin:n {#1}
26   }

```

```

27
28
29 \NewDocumentCommand \tagmccend { }
30 {
31   \tag_mc_end:
32 }
33
34 \NewDocumentCommand \tagmccuse { m }
35 {
36   \tag_mc_use:n {#1}
37 }
38 </base>

```

(End of definition for \tagmccbegin, \tagmccend, and \tagmccuse. These functions are documented on page 35.)

\tagmccifinTF This is a wrapper around \tag_mc_if_in: and tests if an mc is open or not. It is mostly of importance for pdf_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 \tagmccifinTF { m m }
41 {
42   \tag_mc_if_in:TF { #1 } { #2 }
43 }
44 </package>

```

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

10 Commands for the structure

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

```

45 <*base>
46 \NewDocumentCommand \tagstructbegin { m }
47 {
48   \tag_struct_begin:n {#1}
49 }
50
51 \NewDocumentCommand \tagstructend { }
52 {
53   \tag_struct_end:
54 }
55
56 \NewDocumentCommand \tagstructuse { m }
57 {
58   \tag_struct_use:n {#1}
59 }
60 </base>

```

(End of definition for \tagstructbegin, \tagstructend, and \tagstructuse. These functions are documented on page 36.)

11 Socket support

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

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

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

```

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

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

```

(End of definition for `\tag_socket_use:n`, `\tag_socket_use:nn`, and `\UseTaggingSocket`. These functions are documented on page 38.)

12 Debugging

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

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

```

```

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

```

(End of definition for `\ShowTagging`. This function is documented on page 36.)

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.

```

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

```

(End of definition for `mc-data (show-key)`. This function is documented on page 36.)

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

```

111 \keys_define:nn { __tag / show }
112 { mc-current .code:n =
113   {
114     \bool_if:NTF \g__tag_mode_lua_bool
115     {
116       \sys_if_engine_luatex:T
117         {
118           \int_compare:nNnTF
119             { -2147483647 }
120             =
121             {
122               \lua_now:e
123                 {
124                   tex.print
125                     (tex.getattribute
126                       (luatexbase.attributes.g__tag_mc_cnt_attr))
127                 }
128             }
129           {
130             \lua_now:e
131               {
132                 ltx.__tag.trace.log
133                   (
134                     "mc-current:~no~MC~open,~current~abscnt
135                     =\__tag_get_mc_abs_cnt:"
136                     ,0
137                   )
138                 texio.write_nl("")
139               }
140           }
141         }
142     }
143 }

```

```

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

```

(End of definition for mc-current (show-key). This function is documented on page 36.)

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

```

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

```

```

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

```

(End of definition for mc-marks (show-key). This function is documented on page 36.)

struct-stack_□(show-key)

```

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

```

(End of definition for struct-stack (show-key). This function is documented on page 36.)

debug/structures_□(show-key)

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

```

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

```

```

234         {
235             \prop_map_function:cN
236             {g__tag_struct_debug_##1_prop}
237             \msg_show_item_unbraced:nn
238         }
239         { } { }
240     \msg_term:nneeee
241     { tag/debug } { show-kids }
242     { ##1 }
243     {
244         \seq_map_function:cN
245         {g__tag_struct_debug_kids_##1_seq}
246         \msg_show_item_unbraced:n
247     }
248     { } { }
249 }
250 }
251 ,debug/structures .default:n = 0
252 }
253 </debug>

```

(End of definition for `debug/structures (show-key)`. This function is documented on page 36.)

13 Commands to extend document commands

The following commands and code parts are not core commands of tagpdf. They either provide work-arounds for missing functionality elsewhere, or do a first step to apply tagpdf commands to document commands. This part should be regularly revisited to check if the code should go to a better place or can be improved.

```
254 <*package>
```

13.1 Document structure

```

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

```

(End of definition for `\g__tag_root_default_tl`, `activate (setup-key)`, and `activate/socket (setup-key)`. These functions are documented on page 35.)

13.2 Structure destinations

Since TeXlive 2022 pdfTeX and LuaTeX offer support for structure destinations and the pdfmanagement has backend support for. We activate them if structures are actually created. Structure destinations are actually PDF 2.0 only but they don't harm in older PDF and can improve HTML export.

```

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

```

13.3 Fake space

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

```

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

```

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

13.4 Paratagging

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

```

\l__tag_para_bool      At first some variables.
\l__tag_para_flattened_bool 292 </package>
\l__tag_para_show_bool 293 <base>\bool_new:N \l__tag_para_flattened_bool
\g__tag_para_begin_int 294 <base>\bool_new:N \l__tag_para_bool
\g__tag_para_end_int    295 <*package>
\g__tag_para_main_begin_int 296 \int_new:N \g__tag_para_begin_int
\g__tag_para_main_end_int 297 \int_new:N \g__tag_para_end_int
\g__tag_para_main_struct_tl 298 \int_new:N \g__tag_para_main_begin_int
\l__tag_para_tag_default_tl 299 \int_new:N \g__tag_para_main_end_int
\l__tag_para_tag_tl
\l__tag_para_main_tag_tl
\l__tag_para_attr_class_tl
\l__tag_para_main_attr_class_tl

```

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

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

this is perhaps already defined by the block code

```
307 \tl_if_exist:NF \l__tag_para_attr_class_tl
308 { \tl_new:N \l__tag_para_attr_class_tl }
309 \tl_new:N \l__tag_para_main_attr_class_tl
```

(End of definition for \l__tag_para_bool and others.)

`__tag_gincr_para_main_begin_int:` The global para counter should be set through commands so that `\tag_stop:` can stop them.

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

(End of definition for __tag_gincr_para_main_begin_int: and others.)

`__tag_start_para_ints:`

`__tag_stop_para_ints:`

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

```

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

```

(End of definition for `__tag_start_para_ints:` and `__tag_stop_para_ints:`.)

We want to be able to inspect the current para main structure, so we need a command to store its structure number

`__tag_para_main_store_struct:`

```

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

```

(End of definition for `__tag_para_main_store_struct:`.)

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

```

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

```

and one temporary adaptations for the block module:

```

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

```

`para/tagging` (setup-key) These keys enable/disable locally paratagging. Paragraphs are typically tagged with two structure: A main structure around the whole paragraph, and inner structures around the various chunks. Debugging can be activated locally with `debug/show=para`, this can affect the typesetting as the small numbers are boxes and they have a (small) height. Debugging can be deactivated with `debug/show=paraOff`. The `para/tag` key sets the tag used by the inner structure, `para/maintag` the tag of the outer structure, both can also be changed with `\tag_tool:n`

```

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

```

the deprecated names

```

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

```

(End of definition for `para/tagging` (setup-key) and others. These functions are documented on page 37.)

Helper command for debugging:

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

By default we assign the plain plug:

```

507 \socket_assign_plug:nn { tag-support/para/begin}{plain}
508 \socket_assign_plug:nn { tag-support/para/end}{plain}

```

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

```

509 \AddToHook{para/begin}{ \socket_use:n { tag-support/para/begin }
510 }
511 \AddToHook{para/end} { \socket_use:n { tag-support/para/end } }

```

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

```

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

```

```

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

```

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.

```

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

```

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

```

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

```

```

565         { \_tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
566     \tl_put_right:Nn \@kernel@before@cc1v
567     {
568         \_tag_check_typeout_v:n {====>-In~\token_to_str:N \@makecol\c_space_tl\the\c@p
569         \_tag_add_missing_mcs_to_stream:Nn \@cc1v {main}
570     }
571     \tl_put_right:Nn \@kernel@tagsupport@@makecol
572     {
573         \_tag_check_typeout_v:n {====>-In~\token_to_str:N \@makecol\c_space_tl\the\c@p
574         \_tag_add_missing_mcs_to_stream:Nn \@outputbox {main}
575     }
576     \tl_put_right:Nn \@mult@ptagging@hook
577     {
578         \_tag_check_typeout_v:n {====>-In~\string\page@sofar}
579         \process@cols\mult@firstbox
580         {
581             \_tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
582         }
583         \_tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
584     }
585 }
586 }
587 }
588 \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}`

```

589 \newcommand\tagpdfparaOn {}
590 \newcommand\tagpdfparaOff{}
591 \end{package}
592 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
593 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End of definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 37.)

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

```

\@hangfrom
{
  \tagstructbegin{tag=H1}%
  \tagmcbegin {tag=H1}%
  #2
}
{#3\tagpdfsuppressmarks{\tagmcbegin}\tagstructend}%
594 \NewDocumentCommand\tagpdfsuppressmarks{m}
595   {{\use:c{__tag_mc_disable_marks:} #1}}

```

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

13.5 Header and footer

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

```

596 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
597 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
598 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
599 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
600
601 \AddToHook{begindocument}
602 {
603   \cs_if_exist:NT \@kernel@before@head
604   {
605     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head;}
606     \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head;}
607     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot;}
608     \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot;}
609   }
610 }
611
612 \bool_new:N \g__tag_saved_in_mc_bool
613 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
614 {
615   \bool_set_false:N \l__tag_para_bool
616   \bool_if:NTF \g__tag_mode_lua_bool
617   {
618     \tag_mc_end_push:
619   }
620   {
621     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
622     \bool_gset_false:N \g__tag_in_mc_bool
623   }
624   \tag_mc_begin:n {artifact}
625   \tag_stop:n{headfoot}
626 }
627 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
628 {
629   \tag_start:n{headfoot}
630   \tag_mc_end:
631   \bool_if:NTF \g__tag_mode_lua_bool
632   {
633     \tag_mc_begin_pop:n{ }
634   }
635   {
636     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
637   }
638 }

```

This version allows to use an Artifact structure

```

639 \__tag_attr_new_entry:nn {\__tag/attr/pagination}{/0/Artifact/Type/Pagination}
640 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
641 {
642   \bool_set_false:N \l__tag_para_bool

```

```

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

```

And now the keys

`page/exclude-header-footerU(setup-key)`
`exclude-header-footerU(deprecated)`

```

669 \keys_define:nn { __tag / setup }
670 {
671   page/exclude-header-footer .choice:,
672   page/exclude-header-footer / true .code:n =
673   {
674     \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
675     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
676     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
677     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
678   },
679   page/exclude-header-footer / pagination .code:n =
680   {
681     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
682     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
683     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
684     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
685   },
686   page/exclude-header-footer / false .code:n =
687   {
688     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
689     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
690     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
691     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
692   },
693   page/exclude-header-footer .default:n = true,

```

```

694   page/exclude-header-footer .initial:n = true,
deprecated name
695   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
696 }

```

(End of definition for page/exclude-header-footer (setup-key) and exclude-header-footer (deprecated). These functions are documented on page 38.)

13.6 Links

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

```

697 \hook_gput_code:nnn
698   {pdfannot/link/URI/before}
699   {tagpdf}
700   {
701     \tag_mc_end_push:
702     \tag_struct_begin:n { tag=Link }
703     \tag_mc_begin:n { tag=Link }
704     \pdfannot_dict_put:nne
705       { link/URI }
706       { StructParent }
707       { \tag_struct_parent_int: }
708   }
709
710 \hook_gput_code:nnn
711   {pdfannot/link/URI/after}
712   {tagpdf}
713   {
714     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
715     \tag_mc_end:
716     \tag_struct_end:
717     \tag_mc_begin_pop:n{ }
718   }
719
720 \hook_gput_code:nnn
721   {pdfannot/link/GoTo/before}
722   {tagpdf}
723   {
724     \tag_mc_end_push:
725     \tag_struct_begin:n{tag=Link}
726     \tag_mc_begin:n{tag=Link}
727     \pdfannot_dict_put:nne
728       { link/GoTo }
729       { StructParent }
730       { \tag_struct_parent_int: }
731   }
732
733 \hook_gput_code:nnn
734   {pdfannot/link/GoTo/after}
735   {tagpdf}
736   {

```

```
737 \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
738 \tag_mc_end:
739 \tag_struct_end:
740 \tag_mc_begin_pop:n{ }
741
742 }
743
744 % "alternative descriptions " for PAX3. How to get better text here??
745 \pdfannot_dict_put:nnn
746 { link/URI }
747 { Contents }
748 { (url) }
749
750 \pdfannot_dict_put:nnn
751 { link/GoTo }
752 { Contents }
753 { (ref) }
754
</package>
```

Part III

The tagpdf-tree module

Commands trees and main dictionaries

Part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2024-02-29} {0.98x}
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 and OpenAction

The StructTreeRoot and the MarkInfo entry must be added to the catalog. If there is an OpenAction entry we must update it, so that it contains also a structure destination. We do it late so that we can win, but before the pdfmanagement hook.

```

__tag/struct/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.)

```

```

\g__tag_tree_openaction_struct_tl We need a variable that indicates which structure is wanted in the OpenAction. By
default we use 1 (the Document structure).
30 \tl_new:N \g__tag_tree_openaction_struct_tl
31 \tl_gset:Nn \g__tag_tree_openaction_struct_tl {1}
(End of definition for \g__tag_tree_openaction_struct_tl.)

```

viewer/startpage_ (setup-key) We also need an option to setup the start structure. So we setup a key which sets the variable to the current structure. This still requires hyperref to do most of the job, this should perhaps be changed.

```

32 \keys_define:nn { __tag / setup }
33 {
34   viewer/startstructure .code:n =
35   {
36     \tl_gset:Ne \g__tag_tree_openaction_struct_tl {#1}
37   }
38   ,viewer/startstructure .default:n = { \int_use:N \c@g__tag_struct_abs_int }
39 }

```

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

The OpenAction should only be updated if it is there. So we inspect the Catalog-prop:

```

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

```

we only do something if the OpenAction is an array (as set by hyperref) in other cases we hope that the author knows what they did.

```

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

```

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

```

56             \int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1}
57             { /\seq_item:Nn\l__tag_tmpa_seq{2} }
58             { ] }
59         >>
60     }
61 }
62 }
63 }
64 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
65 {
66     \bool_if:NT \g__tag_active_tree_bool
67     {
68         \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
69         \pdfmanagement_add:nne
70         { Catalog }
71         { StructTreeRoot }
72         { \pdf_object_ref:n { __tag/struct/0 } }
73         \__tag_tree_update_openaction:
74     }
75 }

```

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`

```

76 \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

```

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

```

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.

```

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

```

```

93     {
94     \__tag_struct_get_id:n{##1}~\pdf_object_ref:n{__tag/struct/##1}~
95     }
96 \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
97     {
98     \pdf_object_unnamed_write:ne {dict}
99     { /Limits~[\__tag_struct_get_id:n{##1}~\l__tag_tmpa_int+1}~\__tag_struct_get_id:n{##1}~
100     /Names~[\l__tag_tmpa_tl]
101     }
102     \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
103     \int_zero:N \l__tag_tmpa_int
104     \tl_clear:N \l__tag_tmpa_tl
105     }
106   }
107 \tl_if_empty:NF \l__tag_tmpa_tl
108   {
109   \pdf_object_unnamed_write:ne {dict}
110   {
111   /Limits~
112   [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int-\l__tag_tmpa_int+1}~
113   \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
114   /Names~[\l__tag_tmpa_tl]
115   }
116   \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
117   }
118 \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
119 \__tag_prop_gput:cne
120   { g__tag_struct_0_prop }
121   { IDTree }
122   { \pdf_object_ref_last: }
123 }

```

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.

```

124 \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
125   {
126   \__tag_prop_gput:cne
127     { g__tag_struct_0_prop }
128     { ParentTree }
129     { \pdf_object_ref:n { __tag/tree/parenttree } }
130   \__tag_prop_gput:cne
131     { g__tag_struct_0_prop }
132     { RoleMap }
133     { \pdf_object_ref:n { __tag/tree/rolemap } }
134   \__tag_struct_fill_kid_key:n { 0 }
135   \__tag_struct_get_dict_content:nN { 0 } \l__tag_tmpa_tl
136   \pdf_object_write:nne
137     { __tag/struct/0 }
138     {dict}
139     {
140     \l__tag_tmpa_tl

```

```

141     }
142   }

```

(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.

```

143 \cs_new_protected:Npn \_tag_tree_write_structelements:
144   {
145     \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
146     {
147       \_tag_struct_write_obj:n { ##1 }
148     }
149   }

```

(End of definition for `_tag_tree_write_structelements:`.)

1.5 ParentTree

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

```

150 \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`.

```

151 \newcounter { g__tag_parenttree_obj_int }
152 \hook_gput_code:nnn{begindocument}{tagpdf}
153   {
154     \int_gset:Nn
155       \c@g__tag_parenttree_obj_int
156       { \_tag_property_ref_lastpage:nn{abspage}{100} }
157   }

```

(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`

```

158 \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.

```

159 \cs_new_protected:Npn \_tag_parenttree_add_objr:nn #1 #2 % #1 StructParent number, #2 objref
160   {
161     \tl_gput_right:Ne \g__tag_parenttree_objr_tl
162     {
163       #1 \c_space_tl #2 ^^J
164     }
165   }

```

(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.

```
166 \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.

```
167 \cs_new_protected:Npn \_tag_tree_parenttree_rerun_msg: {}
```

```
168 \cs_new_protected:Npn \_tag_tree_fill_parenttree:
```

```
169 {
```

```
170   \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clear
```

```
171   { %page ##1
```

```
172     \prop_clear:N \l__tag_tmpa_prop
```

```
173     \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{tagmcabs}{-1}}
```

```
174     {
```

```
175       %mcid###1
```

```
176       \int_compare:nT
```

```
177         {\_tag_property_ref:enn{mcid-###1}{tagabspage}{-1}=##1} %mcid is on current page
```

```
178         {% yes
```

```
179           \prop_put:Nee
```

```
180             \l__tag_tmpa_prop
```

```
181             {\_tag_property_ref:enn{mcid-###1}{tagmcid}{-1}}
```

```
182             {\prop_item:Nn \g__tag_mc_parenttree_prop {###1}}
```

```
183           }
```

```
184         }
```

```
185     \tl_put_right:Ne\l__tag_parenttree_content_tl
```

```
186     {
```

```
187       \int_eval:n {##1-1}\c_space_tl
```

```
188       [\c_space_tl %]
```

```
189     }
```

```
190   \int_step_inline:nnnn
```

```
191   {0}
```

```
192   {1}
```

```
193   { \prop_count:N \l__tag_tmpa_prop -1 }
```

```
194   {
```

```
195     \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
```

```
196     {% page#1:mcid##1:\l__tag_tmpa_tl :content
```

```
197     \tl_put_right:Ne \l__tag_parenttree_content_tl
```

```
198     {
```

```
199       \pdf_object_if_exist:eTF { __tag/struct/\l__tag_tmpa_tl }
```

```
200       {
```

```
201         \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
```

```
202       }
```

```
203       {
```

```
204         null
```

```
205       }
```

```
206       \c_space_tl
```

```
207     }
```

```
208   }
```

```
209   {
```

```
210     \cs_set_protected:Npn \_tag_tree_parenttree_rerun_msg:
```

```
211     {
```

```

212             \msg_warning:nn { tag } {tree-mcid-index-wrong}
213         }
214     }
215 }
216 \tl_put_right:Nn
217   \l__tag_parenttree_content_tl
218   {%[
219     ]^^J
220   }
221 }
222 }

```

(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.

```

223 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
224 {
225   \tl_set:Nn \l__tag_parenttree_content_tl
226     {
227     \lua_now:e
228     {
229       ltx.__tag.func.output_parenttree
230       (
231         \int_use:N\g_shipout_readonly_int
232       )
233     }
234   }
235 }

```

(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?

```

236 \cs_new_protected:Npn \__tag_tree_write_parenttree:
237 {
238   \bool_if:NTF \g__tag_mode_lua_bool
239     {
240     \__tag_tree_lua_fill_parenttree:
241     }
242   {
243     \__tag_tree_fill_parenttree:
244   }
245   \__tag_tree_parenttree_rerun_msg:
246   \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
247   \pdf_object_write:nne { __tag/tree/parenttree }{dict}
248   {
249     /Nums\c_space_tl [\l__tag_parenttree_content_tl]
250   }
251 }

```

(End of definition for __tag_tree_write_parenttree:.)

1.6 Rolemap dictionary

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

```
__tag/tree/rolemap At first we reserve again an object. Rolemap is also used in PDF 2.0 as a fallback.  
252 \pdf_object_new:n { __tag/tree/rolemap }  
(End of definition for __tag/tree/rolemap.)
```

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

```
253 \cs_new_protected:Npn \__tag_tree_write_rolemap:  
254 {  
255   \bool_if:NT \g__tag_role_add_mathml_bool  
256   {  
257     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop  
258     {  
259       \prop_gput:Nnn \g__tag_role_rolemap_prop {##1}{Span}  
260     }  
261   }  
262   \prop_map_inline:Nn \g__tag_role_rolemap_prop  
263   {  
264     \tl_if_eq:nnF {##1}{##2}  
265     {  
266       \pdfdict_gput:nne {g__tag_role/RoleMap_dict}  
267       {##1}  
268       {\pdf_name_from_unicode_e:n{##2}}  
269     }  
270   }  
271   \pdf_object_write:nne { __tag/tree/rolemap }{dict}  
272   {  
273     \pdfdict_use:n{g__tag_role/RoleMap_dict}  
274   }  
275 }
```

(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:  
276 \cs_new_protected:Npn \__tag_tree_write_classmap:  
277 {  
278   \tl_clear:N \l__tag_tmpa_tl  
279   \seq_gremove_duplicates:N \g__tag_attr_class_used_seq  
280   \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq  
281   {  
282     ##1\c_space_tl  
283     <<  
284     \prop_item:Nn  
285     \g__tag_attr_entries_prop
```

```

286         {##1}
287     >>
288     }
289     \tl_set:Nc \l__tag_tmpa_tl
290     {
291         \seq_use:Nn
292             \l__tag_tmpa_seq
293             { \iow_newline: }
294     }
295     \tl_if_empty:NF
296         \l__tag_tmpa_tl
297     {
298         \pdf_object_new:n { __tag/tree/classmap }
299         \pdf_object_write:nne
300             { __tag/tree/classmap }
301             {dict}
302             { \l__tag_tmpa_tl }
303         \__tag_prop_gput:cne
304             { g__tag_struct_0_prop }
305             { ClassMap }
306             { \pdf_object_ref:n { __tag/tree/classmap } }
307     }
308 }

```

(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

```

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

```

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

__tag_tree_write_namespaces:

```

310 \cs_new_protected:Npn \__tag_tree_write_namespaces:
311     {
312         \pdf_version_compare:NnF < {2.0}
313         {
314             \prop_map_inline:Nn \g__tag_role_NS_prop
315             {
316                 \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
317                 {
318                     \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
319                     {
320                         \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
321                     }
322                     \pdfdict_gput:nne{g__tag_role/namespace_##1_dict}
323                     {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
324                 }
325                 \pdf_object_write:nne{tag/NS/##1}{dict}
326                 {
327                     \pdfdict_use:n {g__tag_role/namespace_##1_dict}

```

```

328     }
329   }
330   \pdf_object_write:nne {__tag/tree/namespaces}{array}
331   {
332     \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
333   }
334 }
335 }

```

(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:`

```

336 \hook_new:n {tagpdf/finish/before}
337 \cs_new_protected:Npn \__tag_finish_structure:
338   {
339     \bool_if:NT\g__tag_active_tree_bool
340     {
341       \hook_use:n {tagpdf/finish/before}
342       \__tag_tree_final_checks:
343       \__tag_tree_write_parenttree:
344       \__tag_tree_write_idtree:
345       \__tag_tree_write_rolemap:
346       \__tag_tree_write_classmap:
347       \__tag_tree_write_namespaces:
348       \__tag_tree_write_structelements: %this is rather slow!!
349       \__tag_tree_write_structtreeroot:
350     }
351   }

```

(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.

```

352 \hook_gput_code:nnn{begindocument}{tagpdf}
353   {
354     \bool_if:NT\g__tag_active_tree_bool
355     {
356       \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
357       {
358         \pdfmanagement_add:nne
359         { Page }
360         { StructParents }
361         { \int_eval:n { \g_shipout_readonly_int } }
362       }
363     }
364   }
365 </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.

`\tag_mc_reset_box:N` \star `\tag_mc_reset_box:N` $\{(box)\}$

New: 2023-06-11

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

2 Public keys

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

`tag□(mc-key)`

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

`artifact□(mc-key)`

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

`raw□(mc-key)`

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

`alt□(mc-key)`

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

`actualtext□(mc-key)`

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

`label□(mc-key)`

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

`stash□(mc-key)`

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

The code is 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} {2024-02-29} {0.98x}
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_in_mc_bool`

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

```

16 \bool_new:N \g__tag_in_mc_bool

```

(End of definition for \g__tag_in_mc_bool.)

`\g__tag_mc_parenttree_prop`

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

key: absolute number of the mc (tagmcabs)
value: the structure number the mc is in

```

17 \__tag_prop_new_linked:N \g__tag_mc_parenttree_prop

```

(End of definition for \g__tag_mc_parenttree_prop.)

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

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

(End of definition for `\g__tag_mc_parenttree_prop`.)

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

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

(End of definition for `\l__tag_mc_artifact_type_tl`.)

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

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

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

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

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

```
\g__tag_mc_key_tag_tl
```

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

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

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

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

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

3.2 Functions

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

tagabspage: the absolute page, `\g_shipout_readonly_int`,

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

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

```
27 {
```

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

```
29 }
```

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

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

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

```
31 {
```

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

```
33 }
```

```
34 </shared>
```

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

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

TODO: is testing for struct the right test?

```

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

```

(End of definition for `\tag_mc_use:n`. This function is documented on page 68.)

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

```

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

```

(End of definition for `\tag_mc_artifact_group_begin:n` and `\tag_mc_artifact_group_end:.` These functions are documented on page 68.)

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

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

(End of definition for `\tag_mc_reset_box:N`. This function is documented on page 69.)

`\tag_mc_end_push:`

`\tag_mc_begin_pop:n`

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

(End of definition for `\tag_mc_end_push:` and `\tag_mc_begin_pop:n`. These functions are documented on page 68.)

3.3 Keys

This are the keys where the code can be shared between the modes.

`stash_(mc-key)` the two internal artifact keys are use to define the public artifact. For now we add support for the subtypes Header and Footer. Watermark,PageNum, LineNum,Redaction,Bates will be added if some use case emerges. If some use case for /BBox and /Attached emerges, it will be perhaps necessary to adapt the code.

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

(End of definition for `stash (mc-key)`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 69.)

```
159 </shared>
```

Part V

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

1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2024-02-29} {0.98x}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2024-02-29} {0.98x}
8 {part of tagpdf - debugging code related to marking chunks - generic mode}
9 </debug>
```

1.1 Variables

```
10 <*generic>
```

`\l__tag_mc_ref_abbrev_t1` 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 tagabbrev attribute retrieved from a label.

```
11 \tl_new:N \l__tag_mc_ref_abbrev_t1
```

(End of definition for `\l__tag_mc_ref_abbrev_t1`.)

`\l__tag_mc_tmpa_t1` temporary variable

```
12 \tl_new:N \l__tag_mc_tmpa_t1
```

(End of definition for `\l__tag_mc_tmpa_t1`.)

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

```
13 \newmarks \g__tag_mc_marks
```

(End of definition for `\g__tag_mc_marks`.)

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

`\g__tag_mc_footnote_marks_seq`

`\g__tag_mc_multicol_marks_seq`

```
14 \seq_new:N \g__tag_mc_main_marks_seq
```

```
15 \seq_new:N \g__tag_mc_footnote_marks_seq
```

```
16 \seq_new:N \g__tag_mc_multicol_marks_seq
```

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

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

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

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

1.2 Functions

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

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

```

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

```

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

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

```

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

```

(End of definition for __tag_mc_disable_marks:.)

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

```

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

```

(End of definition for __tag_mc_get_marks:.)

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

```

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

```

```

94     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_d
95   }
96   {
97     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
98   }
99   \prop_gput:Nee \g__tag_mc_parenttree_prop
100   {#2}
101   {#3}
102 }
103 \cs_generate_variant:Nn \__tag_mc_store:nnn {eee}

```

(End of definition for __tag_mc_store:nnn.)

__tag_mc_insert_extra_tmb:n These two functions should be used in the output routine at the place where a mc-literal could be missing due to a page break or some other split. They check (with the help of the marks) if a extra-tmb or extra-tme is needed. The tmb command stores also the mc into the structure, the tme has to store the data for a following extra-tmb. The argument takes a stream name like main or footnote to allow different handling there. The content of the marks must be stored before (with \@@_mc_get_marks: or manually) into \l_@@_mc_firstmarks_seq and \l_@@_mc_botmarks_seq so that the tests can use them.

```

104 \cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
105   {
106     \__tag_check_typeout_v:n {=>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
107     \__tag_check_typeout_v:n {=>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
108     \__tag_check_if_mc_tmb_missing:TF
109     {
110       \__tag_check_typeout_v:n {=>~ TMB~ ~ missing~ --- inserted}
111       %test if artifact
112       \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
113         1}
114         {
115           \tl_set:Nc \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
116           \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
117         }
118         {
119           \exp_args:Ne
120           \__tag_mc_bdc_mcid:n
121           {
122             \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
123           }
124           \str_if_eq:eeTF
125           {
126             \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
127           }
128           {}
129           {
130             %store
131             \__tag_mc_store:eee
132             {
133               \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
134             }
135             { \int_eval:n{\c@g__tag_MCID_abs_int} }
136             {

```

```

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

```

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

1.3 Looking at MC marks in boxes

__tag_add_missing_mcs:Nn Assumptions:

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

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

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

```

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

```

```

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

```

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

```

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

```

The box added at the bottom will get the depth of the original box. This way we can arrange that from the outside everything looks as before.

```

175     \box_set_ht:Nn \l__tag_tmpb_box \c_zero_dim
176     \box_set_dp:Nn \l__tag_tmpb_box { \box_dp:N #1 }

```

We need to set `\boxmaxdepth` in case the original box has an unusually large depth, otherwise that depth is not preserved when we string things together.

```

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

```

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

```

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

```

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

```

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

```

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

`__tag_add_missing_mcs_to_stream:Nn`

This is the main command to add mc to the stream. It is therefore guarded by the mc-boolean.

If we aren't in the main stream then processing is a bit more complicated because to get at the marks in the box we need to artificially split it and then look at the split marks.

First argument is the box to update and the second is the "stream". In lua mode the command is a no-op.

```

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

```

First set up a temp box for trial splitting.

```

188     \vbadness\maxdimen
189     \box_set_eq:NN \l__tag_tmpa_box #1

```

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

```

190     \vbox_set_split_to_ht:NNn \l__tag_tmpa_box \l__tag_tmpa_box \c_max_dim

```

As a side effect of this split we should now have the first and bottom split marks set up. We use this to set up `\l__tag_mc_firstmarks_seq`

```

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

```

Some debugging info:

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

If this mark was empty then clearly the bottom mark will too be empty. Thus in this case we make use of the saved bot mark from the previous chunk. Note that if this is the first chunk in the stream the global seq would contain a random value, but then we can't end in this branch because the basis assumption is that streams are properly marked up so the first chunk would always have a mark at the beginning!

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

We also update the bot mark to the same value so that we can later apply `__tag_add_missing_mcs:Nn` with the data structures in place (see assumptions made there).

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

If there was a first mark then there is also a bot mark (and it can't be the same as our marks always come in pairs). So if that branch is chosen we update `\l__tag_mc_botmarks_seq` from the bot mark.

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

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

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

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

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

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

```

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

```

(End of definition for __tag_mc_if_in:TF and \tag_mc_if_in:TF. This function is documented on page 68.)

__tag_mc_bmc:n These are the low-level commands. There are now equal to the pdfmanagement commands generic mode, but we use an indirection in case luamode need something else.
 __tag_mc_emc: change 04.08.2018: the commands do not check the validity of the arguments or try to
 __tag_mc_bdc:nn change 2023-08-18: we are delaying the writing to the shipout.

```

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

```

(End of definition for __tag_mc_bmc:n, __tag_mc_emc:, and __tag_mc_bdc:nn.)

__tag_mc_bdc_mcid:nn This create a BDC mark with an /MCID key. Most of the work here is to get the current
 __tag_mc_bdc_mcid:n number value for the MCID: they must be numbered by page starting with 0 and then
 __tag_mc_handle_mcid:nn successively. The first argument is the tag, e.g. P or Span, the second is used to pass
 __tag_mc_handle_mcid:VV more properties. Starting with texlive 2023 this is much simpler and faster as we can
 use delay the numbering to the shipout. We also define a wrapper around the low-level
 command as luamode will need something different.

```

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

```

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

```

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

```

```

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

```

```

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

```

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

__tag_mc_handle_stash:n This is the handler which puts a mc into the the current structure. The argument is the number of the mc. Beside storing the mc into the structure, it also has to record the structure for the parent tree. The name is a bit confusing, it does *not* handle mc with the stash key TODO: why does luamode use it for begin + use, but generic mode only for begin?

```

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

```

(End of definition for __tag_mc_handle_stash:n.)

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

```

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

```

(End of definition for `_tag_mc_bmc_artifact:`, `_tag_mc_bmc_artifact:n`, and `_tag_mc_handle_artifact:N`.)

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

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

(End of definition for `_tag_get_data_mc_tag:.`)

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

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

set default MC tags to structure:

```
370   \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
371   \tl_gset_eq:NN \g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
372   \keys_set:nn { \_tag / mc } {#1}
373   \bool_if:NTF \l__tag_mc_artifact_bool
374   { %handle artifact
375     \_tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
376     \exp_args:NV
377     \_tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
378   }
379   { %handle mcid type
380     \_tag_check_mc_tag:N \l__tag_mc_key_tag_tl
381     \_tag_mc_handle_mcid:VV
382       \l__tag_mc_key_tag_tl
383       \l__tag_mc_key_properties_tl
384     \_tag_mc_begin_marks:oof{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
385     \tl_if_empty:NF {\l__tag_mc_key_label_tl}
386     {
387       \exp_args:NV
388       \_tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl
389     }
```

```

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

```

```

444     \_tag_mc_end_marks:
445     }
446 (*debug)
447     {
448     \_tag_debug_mc_end_ignore:
449     }
450 </debug>
451     }
452 </generic | debug>

```

(End of definition for `\tag_mc_begin:n` and `\tag_mc_end:.` These functions are documented on page 68.)

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

```

```

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

```

(End of definition for tag (mc-key) and others. These functions are documented on page 69.)

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

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

`tag` : the type (a string)

`raw` : more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2024-02-29} {0.98x}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2024-02-29} {0.98x}
8   {part of tagpdf - debugging code related to marking chunks - lua mode}
9 </debug>
```

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

```

10 (*luamode)
11 \hook_gput_code:nnn{begindocument}{tagpdf/mc}
12 {
13   \bool_if:NT\g__tag_active_space_bool
14   {
15     \lua_now:e
16     {
17       if~luatexbase.callbacktypes.pre_shipout_filter~then~
18         luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
19           ltx.__tag.func.space_chars_shipout(TAGBOX)~return~true~
20           end, "tagpdf")~
21       if~luatexbase.declare_callback_rule~then~
22         luatexbase.declare_callback_rule("pre_shipout_filter", "luaotfload.dvi", "aft
23       end~
24     end
25   }
26   \lua_now:e
27   {
28     if~luatexbase.callbacktypes.pre_shipout_filter~then~
29       token.get_next()~
30     end
31     }@secondoftwo@gobble
32     {
33       \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
34       {
35         \lua_now:e
36         { ltx.__tag.func.space_chars_shipout (tex.box["ShipoutBox"]) }
37       }
38     }
39   }
40   \bool_if:NT\g__tag_active_mc_bool
41   {
42     \lua_now:e
43     {
44       if~luatexbase.callbacktypes.pre_shipout_filter~then~
45         luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
46           ltx.__tag.func.mark_shipout(TAGBOX)~return~true~
47           end, "tagpdf")~
48       end
49     }
50     \lua_now:e
51     {
52       if~luatexbase.callbacktypes.pre_shipout_filter~then~
53         token.get_next()~
54       end
55       }@secondoftwo@gobble
56       {
57         \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
58         {
59           \lua_now:e
60           { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
61         }

```

```

62     }
63   }
64 }

```

1.1 Commands

`_tag_add_missing_mcs_to_stream:Nn` This command is used in the output routine by the ptagging code. It should do nothing in luamode.

```

65 \cs_new_protected:Npn \_tag_add_missing_mcs_to_stream:Nn #1#2 {}

```

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

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

```

\underline{\_tag_mc_if_in:TF} 66 \prg_new_conditional:Nnn \_tag_mc_if_in: {p,T,F,TF}
\underline{\_tag_mc_if_in_p:} 67 {
\underline{\_tag_mc_if_in:TF} 68   \int_compare:nNnTF
69     { -2147483647 }
70     =
71     {\lua_now:e
72       {
73         tex.print(\int_use:N \c_document_cctab,tex.getattribute(luatexbase.attributes.g__t
74       )
75     }
76     { \prg_return_false: }
77     { \prg_return_true: }
78   }
79
80 \prg_new_eq_conditional:NNn \tag_mc_if_in: \_tag_mc_if_in: {p,T,F,TF}

```

(End of definition for `_tag_mc_if_in:TF` and `\tag_mc_if_in:TF`. This function is documented on page 68.)

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

```

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

```

```

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

```

(End of definition for __tag_mc_lua_set_mc_type_attr:n and __tag_mc_lua_unset_mc_type_attr:.)

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

```

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

```

(End of definition for __tag_mc_insert_mcid_kids:n and __tag_mc_insert_mcid_single_kids:n.)

__tag_mc_handle_stash:n This is the lua variant for the command to put an mcid absolute number in the current structure.

```

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

```

```

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

```

(End of definition for __tag_mc_handle_stash:n.)

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

```

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

```

set the default tag to the structure:

```

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

```

```

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

```

(End of definition for \tag_mc_begin:n. This function is documented on page 68.)

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

```

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

```

(End of definition for `\tag_mc_end:`. This function is documented on page 68.)

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

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

(End of definition for `\tag_mc_reset_box:N`. This function is documented on page 69.)

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

```
254 \cs_new:Npn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
```

(End of definition for `__tag_get_data_mc_tag:`.)

1.2 Key definitions

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

```

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

```

```

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

```

(End of definition for tag (mc-key) and others. These functions are documented on page 69.)

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>
<code>\tag_struct_use_num:n</code>	<code>\tag_struct_use_num:n{<structure number>}</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.

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

This is a command that allows to update the data of a structure. This often can't be done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the only keyword is `ref` which updates the Ref key (an array)

2 Public keys

2.1 Keys for the structure commands

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

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

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

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

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

`alt_<struct-key>` This key inserts an `/Alt` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.

actualtext_□(struct-key) This key inserts an `/ActualText` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.

lang_□(struct-key) This key allows to set the language for a structure element. The value should be a bcp-identifier, e.g. `de-De`.

ref_□(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>`
AFref_□(struct-key) `AFref = <object reference>`
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
texsource `<object name>` should be the name of an object pointing to the `/Filespec` dictionary
mathml 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.

`texsource` is a special variant of `AF-inline-o` which embeds the file as `.tex` source with the `/AFrelationship` key set to `/Source`. It also sets the `/Desc` key to a (currently) fix text.

`mathml` is a special variant of `AF-inline-o` which embeds the file as `.xml` file with the `/AFrelationship` key set to `/Supplement`. It also sets the `/Desc` key to a (currently) fix text.

The argument of `AF` is an object name referring an embedded file as declared for example with `\pdf_object_new:n` or with the `l3pdffile` module. `AF` expands its argument (this allows e.g. to use some variable for automatic numbering) and can be used more than once, to associate more than one file.

The argument of `AFref` is an object reference to an embedded file or a variable expanding to such a object reference in the format as you would get e.g. from `\pdf_object_ref_last:` or `\pdf_object_ref:n` (and which is different for the various engines!). The key allows to make use of anonymous objects. Like `AF` the `AFref` key expands its argument and can be used more than once, to associate more than one file. *It does not check if the reference is valid!*

The inline keys can be used only once per structure. Additional calls are ignored.

attribute_□(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

role/new-attribute_□(setup-key) `role/new-attribute = {<name>}{<Content>}`
newattribute_□(deprecated)

This key can be used in the setup command `\tagpdfsetup` and allow to declare a new attribute, which can be used as attribute or attribute class. The value are two brace groups, the first contains the name, the second the content.

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2024-02-29} {0.98x}
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`.)

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

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

(End of definition for `\c__tag_struct_null_tl`.)

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

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

(End of definition for `\g__tag_struct_cont_mc_prop`.)

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

```
12 \seq_new:N \g__tag_struct_stack_seq
13 \seq_gpush:Nn \g__tag_struct_stack_seq {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.

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

(End of definition for `\g__tag_struct_tag_stack_seq`.)

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

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

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

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

The data of the StructTreeRoot and the StructElem are in properties: `\g_@@_struct_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)

These seq contain the keys we support in the two object types. They are currently no longer used, but are provided as documentation and for potential future checks. They should be adapted if there are changes in the PDF format.

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

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

3.1 Variables used by the keys

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

```
\g__tag_struct_tag_tl
\g__tag_struct_tag_NS_tl
\l__tag_struct_roletag_tl
\g__tag_struct_roletag_NS_tl
```

```

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

(End of definition for \g__tag_struct_tag_tl and others.)

```

```

\l__tag_struct_key_label_tl This will hold the label value.
62 \tl_new:N \l__tag_struct_key_label_tl

(End of definition for \l__tag_struct_key_label_tl.)

```

```

\l__tag_struct_elem_stash_bool This will keep track of the stash status
63 \bool_new:N \l__tag_struct_elem_stash_bool

(End of definition for \l__tag_struct_elem_stash_bool.)

```

3.2 Variables used by tagging code of basic elements

```

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

```

```

64 \</package>
65 \<base>\prop_new_linked:N \g__tag_struct_dest_num_prop
66 \*package>

(End of definition for \g__tag_struct_dest_num_prop.)

```

```

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

```

```

67 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop

(End of definition for \g__tag_struct_ref_by_dest_prop.)

```

4 Commands

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

```

\__tag_struct_output_prop_aux:nn
\__tag_new_output_prop_handler:n
68 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
69 {
70   \prop_if_in:cnT
71     { g__tag_struct_#1_prop }
72     { #2 }
73     {
74       \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
75     }

```

```

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

```

(End of definition for __tag_struct_output_prop_aux:nn and __tag_new_output_prop_handler:n.)

__tag_struct_prop_gput:nnn The structure props must be filled in various places. For this we use a common command which also takes care of the debug package:

```

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

```

(End of definition for __tag_struct_prop_gput:nnn.)

4.1 Initialization of the StructTreeRoot

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

```

96 \*package
97 \tl_gset:Nn \g__tag_struct_stack_current_tl {0}

```

__tag_pdf_name_e:n

```

98 \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
99 \</package>

```

(End of definition for __tag_pdf_name_e:n.)

g__tag_struct_0_prop
g__tag_struct_kids_0_seq

```

100 \*package
101 \__tag_prop_new:c { g__tag_struct_0_prop }
102 \__tag_new_output_prop_handler:n {0}
103 \__tag_seq_new:c { g__tag_struct_kids_0_seq }
104
105 \__tag_struct_prop_gput:nne
106   { 0 }
107   { Type }
108   { \pdf_name_from_unicode_e:n {StructTreeRoot} }

```

```

109
110 \__tag_struct_prop_gput:nne
111   { 0 }
112   { S }
113   { \pdf_name_from_unicode_e:n {StructTreeRoot} }
114
115 \__tag_struct_prop_gput:nne
116   { 0 }
117   { rolemap }
118   { {StructTreeRoot}{pdf} }
119
120 \__tag_struct_prop_gput:nne
121   { 0 }
122   { parentrole }
123   { {StructTreeRoot}{pdf} }
124

```

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

```

125 \pdf_version_compare:NnF < {2.0}
126 {
127   \__tag_struct_prop_gput:nne
128   { 0 }
129   { Namespaces }
130   { \pdf_object_ref:n { __tag/tree/namespaces } }
131 }
132 </package>

```

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

```

133 <debug>\prop_new:c { g__tag_struct_debug_0_prop }
134 <debug>\seq_new:c { g__tag_struct_debug_kids_0_seq }
135 <debug>\prop_gset_eq:cc { g__tag_struct_debug_0_prop }{ g__tag_struct_0_prop }
136 <debug>\prop_gremove:cn { g__tag_struct_debug_0_prop }{Namespaces}

```

(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
137 (*package)
138 \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
139 {
140   (
141     ID.
142     \prg_replicate:nn
143     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
144     { 0 }
145     \int_to_arabic:n { #1 }
146   )
147 }

```

(End of definition for __tag_struct_get_id:n.)

4.3 Filling in the tag info

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

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

(End of definition for `_tag_struct_set_tag_info:nmn`.)

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

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

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

4.4 Handlings kids

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

`_tag_struct_kid_mc_gput_right:nn`
`_tag_struct_kid_mc_gput_right:ne`

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

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

(End of definition for `_tag_struct_kid_mc_gput_right:nn`.)

`_tag_struct_kid_struct_gput_right:nn`
`_tag_struct_kid_struct_gput_right:ee`

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

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

```

226 <debug>\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent str
227 {
228   \__tag_seq_gput_right:ce
229   { g__tag_struct_kids_#1_seq }
230   {
231     \pdf_object_ref:n { __tag/struct/#2 }
232   }
233 <debug>   \seq_gput_right:cn
234 <debug>   { g__tag_struct_debug_kids_#1_seq }
235 <debug>   {
236 <debug>     Struct~#2
237 <debug>   }
238 }
239
240 <package>\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {eee}

```

(End of definition for __tag_struct_kid_struct_gput_right:nn.)

_tag_struct_kid_OBJR_gput_right:nnn
_tag_struct_kid_OBJR_gput_right:eee

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

```

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

```

(End of definition for __tag_struct_kid_OBJR_gput_right:nnn.)

_tag_struct_exchange_kid_command:N
_tag_struct_exchange_kid_command:c

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

```

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

```

```

267 \seq_gpop_left:NN #1 \l__tag_tmpa_tl
268 \regex_replace_once:nnN
269   { \c{\__tag_mc_insert_mcid_kids:n} }
270   { \c{\__tag_mc_insert_mcid_single_kids:n} }
271   \l__tag_tmpa_tl
272 \seq_gput_left:NV #1 \l__tag_tmpa_tl
273 }
274
275 \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.

```

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

```

check if we get null

```

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

```

```

314         {K}
315         {
316             \seq_item:cn
317             {
318                 g__tag_struct_kids_#1_seq
319             }
320             {1}
321         }
322     }
323 }
324 {
325     \__tag_struct_prop_gput:nne
326     {#1}
327     {K}
328     {
329         \seq_item:cn
330         {
331             g__tag_struct_kids_#1_seq
332         }
333         {1}
334     }
335 }
336 } %
337 }
338 { %many kids, use an array
339     \__tag_struct_prop_gput:nne
340     {#1}
341     {K}
342     {
343         [
344             \seq_use:cn
345             {
346                 g__tag_struct_kids_#1_seq
347             }
348             {
349                 \c_space_tl
350             }
351         ]
352     }
353 }
354 }
355

```

(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.

```

356 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
357 {
358     \tl_clear:N #2
359     \seq_map_inline:cn

```

```

360     {
361       c__tag_struct_
362       \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
363       _entries_seq
364     }
365     {
366       \tl_put_right:Ne
367         #2
368         {
369           \prop_if_in:cnT
370             { g__tag_struct_#1_prop }
371             { ##1 }
372             {
373               \c_space_tl/##1-

```

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

```

374           \cs_if_exist_use:cTF {__tag_struct_format_##1:e}
375           {
376             { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } }
377           }
378           {
379             \prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
380           }
381         }
382       }
383     }
384   }

```

(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:

```

385 \cs_new:Nn\__tag_struct_format_Ref:n{[#1]}
386 \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.

```

387 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
388 {
389   \pdf_object_if_exist:nTF { __tag/struct/#1 }
390   {

```

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.

```

391     \prop_get:cnNF { g__tag_struct_#1_prop } {P}\l__tag_tmpb_tl
392     {
393       \prop_gput:cne { g__tag_struct_#1_prop } {P}{\pdf_object_ref:n { __tag/struct/0 }
394       \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
395       \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
396       {
397         \msg_warning:nnee
398           {tag}
399           {struct-orphan}

```

```

400         { #1 }
401         {\seq_count:c{g__tag_struct_kids_#1_seq}}
402     }
403 }
404 \__tag_struct_fill_kid_key:n { #1 }
405 \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
406 \exp_args:Ne
407     \pdf_object_write:nne
408     { __tag/struct/#1 }
409     {dict}
410     {
411         \l__tag_tmpa_tl\c_space_tl
412         /ID-\__tag_struct_get_id:n{#1}
413     }
414 }
415 }
416 {
417     \msg_error:nnn { tag } { struct-no-objnum } { #1}
418 }
419 }

```

(End of definition for __tag_struct_write_obj:n.)

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

Annotations used as structure content must

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

For a link this looks like this

```

\tag_struct_begin:n { tag=Link }
\tag_mc_begin:n { tag=Link }
(1) \pdfannot_dict_put:nne
    { link/URI }
    { StructParent }
    { \int_use:N\c@g_@@_parenttree_obj_int }
<start link> link text <stop link>
(2+3) \@@_struct_insert_annot:nn {obj ref}{parent num}
\tag_mc_end:
\tag_struct_end:

```

```

420 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat.
421                                     %#2 structparent number
422 {
423     \bool_if:NT \g__tag_active_struct_bool
424     {
425         %get the number of the parent structure:
426         \seq_get:NNF
427         \g__tag_struct_stack_seq
428         \l__tag_struct_stack_parent_tmpa_tl

```

```

429     {
430     \msg_error:nn { tag } { struct-faulty-nesting }
431     }
432     %put the obj number of the annot in the kid entry, this also creates
433     %the OBJR object
434     \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
435     \__tag_struct_kid_OBJR_gput_right:eee
436     {
437     \l__tag_struct_stack_parent_tmpa_tl
438     }
439     {
440     #1 %
441     }
442     {
443     \pdf_pageobject_ref:n { \__tag_property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }
444     }
445     % add the parent obj number to the parent tree:
446     \exp_args:Nne
447     \__tag_parenttree_add_objr:nn
448     {
449     #2
450     }
451     {
452     \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
453     }
454     % increase the int:
455     \int_gincr:N \c@g__tag_parenttree_obj_int
456     }
457 }

```

(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**.

```

458 \cs_new:Npn \__tag_get_data_struct_tag:
459 {
460     \exp_args:Ne
461     \tl_tail:n
462     {
463     \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
464     }
465 }

```

(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**.

```

466 \cs_new:Npn \__tag_get_data_struct_id:
467 {
468     \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
469 }
470 </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

```
471 (*base)
472 \cs_new:Npn \_tag_get_data_struct_num:
473 {
474   \g__tag_struct_stack_current_tl
475 }
476 </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.

```
477 (*base)
478 \cs_new:Npn \_tag_get_data_struct_counter:
479 {
480   \int_use:N \c@g__tag_struct_abs_int
481 }
482 </base>
```

(End of definition for _tag_get_data_struct_counter:.)

5 Keys

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

This socket is used by the tag key. It allows to switch between the latex-tabs and the standard tags.

```
483 (*package)
484 \socket_new:nn { tag/struct/tag }{1}
485 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
486 {
487   \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
488   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
489   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
490   \_tag_check_structure_tag:N \g__tag_struct_tag_tl
491 }
492
493 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
494 {
495   \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
496   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
497   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
498   \_tag_role_get:VVNN \g__tag_struct_tag_tl\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl\l__tag_t
499   \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
500   \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
501   \_tag_check_structure_tag:N \g__tag_struct_tag_tl
502 }
503 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}
```

```

label_(struct-key)
stash_(struct-key) 504 \keys_define:nn { __tag / struct }
parent_(struct-key) 505 {
tag_(struct-key) 506 label .tl_set:N = \l__tag_struct_key_label_tl,
title_(struct-key) 507 stash .bool_set:N = \l__tag_struct_elem_stash_bool,
title-o_(struct-key) 508 parent .code:n =
509 {
alt_(struct-key) 510 \bool_lazy_and:nnTF
actualtext_(struct-key) 511 {
lang_(struct-key) 512 \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
ref_(struct-key) 513 }
E_(struct-key) 514 {
515 \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
516 }
517 { \tl_set:Nc \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
518 {
519 \msg_warning:nnee { tag } { struct-unknown }
520 { \int_eval:n {#1} }
521 { parent~key~ignored }
522 }
523 },
524 parent .default:n = {-1},
525 tag .code:n = % S property
526 {
527 \socket_use:nn { tag/struct/tag }{#1}
528 },
529 title .code:n = % T property
530 {
531 \str_set_convert:Nnnn
532 \l__tag_tmpa_str
533 { #1 }
534 { default }
535 { utf16/hex }
536 \__tag_struct_prop_gput:nne
537 { \int_use:N \c@g__tag_struct_abs_int }
538 { T }
539 { <\l__tag_tmpa_str> }
540 },
541 title-o .code:n = % T property
542 {
543 \str_set_convert:Nonn
544 \l__tag_tmpa_str
545 { #1 }
546 { default }
547 { utf16/hex }
548 \__tag_struct_prop_gput:nne
549 { \int_use:N \c@g__tag_struct_abs_int }
550 { T }
551 { <\l__tag_tmpa_str> }
552 },
553 alt .code:n = % Alt property
554 {
555 \tl_if_empty:oF{#1}
556 {

```

```

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

```

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

```

591 ref .code:n      = % ref property
592 {
593   \tl_clear:N\l__tag_tmpa_tl
594   \clist_map_inline:on {#1}
595   {
596     \tl_put_right:Ne \l__tag_tmpa_tl
597     {~\__tag_property_ref:en{tagpdfstruct-##1}{tagstructobj} }
598   }
599   \__tag_struct_gput_data_ref:ee
600   { \int_use:N \c@g__tag_struct_abs_int } {\l__tag_tmpa_tl}
601 },
602 E .code:n      = % E property
603 {
604   \str_set_convert:Nnon
605   \l__tag_tmpa_str
606   { #1 }
607   { default }
608   { utf16/hex }
609   \__tag_struct_prop_gput:nne

```

```

610         { \int_use:N \c@g__tag_struct_abs_int }
611         { E }
612         { <\l__tag_tmpa_str> }
613     },
614 }

```

(End of definition for label (struct-key) and others. These functions are documented on page 99.)

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/AFref is an array and can be used more than once, so we store it in a tl. which is expanded. AFinline currently uses the fix extension txt. texsource is a special variant which creates a tex-file, it expects a tl-var as value (e.g. from math grabbing)

This variable is used to number the AF-object names

```

615 \int_new:N\g__tag_struct_AFobj_int

\g__tag_struct_AFobj_int
616 \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
617 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
618 % #1 content, #2 extension
619 {
620     \tl_if_empty:nF{#1}
621     {
622         \group_begin:
623         \int_gincr:N \g__tag_struct_AFobj_int
624         \pdffile_embed_stream:neN
625         {#1}
626         {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
627         \l__tag_tmpa_tl
628         \__tag_struct_add_AF:ee
629         { \int_use:N \c@g__tag_struct_abs_int }
630         { \l__tag_tmpa_tl }
631         \__tag_struct_prop_gput:nne
632         { \int_use:N \c@g__tag_struct_abs_int }
633         { AF }
634         {
635             [
636                 \tl_use:c
637                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
638             ]
639         }
640         \group_end:
641     }
642 }

643
644 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
645 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object reference
646 {
647     \tl_if_exist:cTF
648     {
649         g__tag_struct_#1_AF_tl
650     }
651     {

```

```

652     \tl_gput_right:ce
653     { g__tag_struct_#1_AF_tl }
654     { \c_space_tl #2 }
655   }
656   {
657     \tl_new:c
658     { g__tag_struct_#1_AF_tl }
659     \tl_gset:ce
660     { g__tag_struct_#1_AF_tl }
661     { #2 }
662   }
663 }
664 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
665 \keys_define:nn { __tag / struct }
666 {
667   AF .code:n      = % AF property
668   {
669     \pdf_object_if_exist:eTF {#1}
670     {
671       \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e
672       \__tag_struct_prop_gput:nne
673       { \int_use:N \c@g__tag_struct_abs_int }
674       { AF }
675       {
676         [
677           \tl_use:c
678           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
679         ]
680       }
681     }
682     {
683       % message?
684     }
685   },
686   AFref .code:n   = % AF property
687   {
688     \tl_if_empty:eF {#1}
689     {
690       \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
691       \__tag_struct_prop_gput:nne
692       { \int_use:N \c@g__tag_struct_abs_int }
693       { AF }
694       {
695         [
696           \tl_use:c
697           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
698         ]
699       }
700     }
701   },
702   ,AFinline .code:n =
703   {
704     \__tag_struct_add_inline_AF:nn {#1}{txt}
705   }

```

```

706 ,AFinline-o .code:n =
707   {
708     \__tag_struct_add_inline_AF:on {#1}{txt}
709   }
710 ,texsource .code:n =
711   {
712     \group_begin:
713     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX-source)}
714     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
715     \__tag_struct_add_inline_AF:on {#1}{tex}
716     \group_end:
717   }
718 ,mathml .code:n =
719   {
720     \group_begin:
721     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml-representation)}
722     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
723     \__tag_struct_add_inline_AF:on {#1}{xml}
724     \group_end:
725   }
726 }

```

(End of definition for AF (struct-key) and others. These functions are documented on page 100.)

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!

```

727 \keys_define:nn { __tag / setup }
728   {
729     root-AF .code:n =
730     {
731       \pdf_object_if_exist:nTF {#1}
732       {
733         \__tag_struct_add_AF:ee { 0 }{\pdf_object_ref:n {#1}}
734         \__tag_struct_prop_gput:nne
735         { 0 }
736         { AF }
737         {
738           [
739             \tl_use:c
740             { g__tag_struct_0_AF_tl }
741           ]
742         }
743       }
744     }
745   }
746 },
747 }
748 \</package>

```

(End of definition for root-AF (setup-key). This function is documented on page 101.)

6 User commands

`\tag_struct_begin:n`

`\tag_struct_end:`

```

750 (base)\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
751 (base)\cs_new_protected:Npn \tag_struct_end: {}
752 (base)\cs_new_protected:Npn \tag_struct_end:n {}
753 (*package | debug)
754 (package)\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
755 (debug)\cs_set_protected:Npn \tag_struct_begin:n #1 % #1 key-val
756 {
757 (package)\__tag_check_if_active_struct:T
758 (debug)\__tag_check_if_active_struct:TF
759 {
760   \group_begin:
761   \int_gincr:N \c@g__tag_struct_abs_int
762   \__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
763 (debug)   \prop_new:c { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
764   \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
765   \__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
766 (debug)   \seq_new:c { g__tag_struct_debug_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
767   \exp_args:Ne
768   \pdf_object_new:n
769   { \__tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
770   \__tag_struct_prop_gput:nnn
771   { \int_use:N \c@g__tag_struct_abs_int }
772   { Type }
773   { /StructElem }
774   \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
775   \keys_set:nn { \__tag / struct } { #1 }
776   \__tag_struct_set_tag_info:eVV
777   { \int_use:N \c@g__tag_struct_abs_int }
778   \g__tag_struct_tag_tl
779   \g__tag_struct_tag_NS_tl
780   \__tag_check_structure_has_tag:n { \int_use:N \c@g__tag_struct_abs_int }
781   \tl_if_empty:NF
782   \l__tag_struct_key_label_tl
783   {
784     \__tag_property_record:eV
785     {tagpdfstruct-\l__tag_struct_key_label_tl}
786     \c__tag_property_struct_clist
787   }

```

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

```

788   \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
789   {
790     \seq_get:NNF
791     \g__tag_struct_stack_seq
792     \l__tag_struct_stack_parent_tmpa_tl
793     {
794       \msg_error:nn { tag } { struct-faulty-nesting }
795     }
796   }
797   \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int

```

```

798     \__tag_role_get:VVNN
799     \g__tag_struct_tag_t1
800     \g__tag_struct_tag_NS_t1
801     \l__tag_struct_roletag_t1
802     \l__tag_struct_roletag_NS_t1

```

to target role and role NS

```

803     \__tag_struct_prop_gput:nne
804     { \int_use:N \c@g__tag_struct_abs_int }
805     { rolemap }
806     {
807         {\l__tag_struct_roletag_t1}{\l__tag_struct_roletag_NS_t1}
808     }

```

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!

```

809     \str_case:VnTF \l__tag_struct_roletag_t1
810     {
811         {Part} {}
812         {Div} {}
813         {NonStruct} {}
814     }
815     {
816         \prop_get:cnNT
817         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_t1 _prop }
818         { parentrole }
819         \l__tag_get_tmpc_t1
820         {
821             \__tag_struct_prop_gput:nno
822             { \int_use:N \c@g__tag_struct_abs_int }
823             { parentrole }
824             {
825                 \l__tag_get_tmpc_t1
826             }
827         }
828     }
829     {
830         \__tag_struct_prop_gput:nne
831         { \int_use:N \c@g__tag_struct_abs_int }
832         { parentrole }
833         {
834             {\l__tag_struct_roletag_t1}{\l__tag_struct_roletag_NS_t1}
835         }
836     }
837     \seq_gpush:Ne \g__tag_struct_tag_stack_seq
838     {{\g__tag_struct_tag_t1}{\l__tag_struct_roletag_t1}}
839     \tl_gset:NV \g__tag_struct_stack_current_t1 \c@g__tag_struct_abs_int
840     %\seq_show:N \g__tag_struct_stack_seq
841     \bool_if:NF
842     \l__tag_struct_elem_stash_bool
843     {

```

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!

```
844     \__tag_struct_get_parentrole:eNN
845     { \l__tag_struct_stack_parent_tmpa_tl }
846     \l__tag_get_parent_tmpa_tl
847     \l__tag_get_parent_tmpb_tl
848     \__tag_check_parent_child:VVVVN
849     \l__tag_get_parent_tmpa_tl
850     \l__tag_get_parent_tmpb_tl
851     \g__tag_struct_tag_tl
852     \g__tag_struct_tag_NS_tl
853     \l__tag_parent_child_check_tl
854     \int_compare:nNnT { \l__tag_parent_child_check_tl } < 0
855     {
856         \prop_get:cnN
857         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
858         { S }
859         \l__tag_tmpa_tl
860         \msg_warning:nneee
861         { tag }
862         { role-parent-child }
863         { \l__tag_get_parent_tmpa_tl / \l__tag_get_parent_tmpb_tl }
864         { \g__tag_struct_tag_tl / \g__tag_struct_tag_NS_tl }
865         { not~allowed~
866           (struct~\l__tag_struct_stack_parent_tmpa_tl, ~\l__tag_tmpa_tl
867             \c_space_tl-->~struct~\int_eval:n { \c@g__tag_struct_abs_int } )
868         }
869         \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
870         \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
871         \__tag_role_remap:
872         \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
873         \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
874         \__tag_struct_set_tag_info:eVV
875         { \int_use:N \c@g__tag_struct_abs_int }
876         \g__tag_struct_tag_tl
877         \g__tag_struct_tag_NS_tl
878     }
879
880     Set the Parent.
881
882     \__tag_struct_prop_gput:nne
883     { \int_use:N \c@g__tag_struct_abs_int }
884     { P }
885     {
886         \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
887     }
888
889     %record this structure as kid:
890     %\tl_show:N \g__tag_struct_stack_current_tl
891     %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
892     \__tag_struct_kid_struct_gput_right:ee
893     { \l__tag_struct_stack_parent_tmpa_tl }
894     { \g__tag_struct_stack_current_tl }
895     %\prop_show:c { g__tag_struct_ \g__tag_struct_stack_current_tl _prop }
896     %\seq_show:c { g__tag_struct_kids_ \l__tag_struct_stack_parent_tmpa_tl _seq }
897 }
```

the debug mode stores in second prop and replaces value with more suitable ones. (If the structure is updated later this gets perhaps lost, but well ...) This must be done outside of the stash boolean.

```

894 (debug)          \prop_gset_eq:cc
895 (debug)          { g__tag_struct_debug_int_eval:n {\c@g__tag_struct_abs_int}_prop }
896 (debug)          { g__tag_struct_int_eval:n {\c@g__tag_struct_abs_int}_prop }
897 (debug)          \prop_gput:cne
898 (debug)          { g__tag_struct_debug_int_eval:n {\c@g__tag_struct_abs_int}_prop }
899 (debug)          { P }
900 (debug)          {
901 (debug)          \bool_if:NTF \l__tag_struct_elem_stash_bool
902 (debug)          {no-parent:~stashed}
903 (debug)          {
904 (debug)          parent~structure:~\l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
905 (debug)          \l__tag_get_parent_tmpa_tl
906 (debug)          }
907 (debug)          }
908 (debug)          \prop_gput:cne
909 (debug)          { g__tag_struct_debug_int_eval:n {\c@g__tag_struct_abs_int}_prop }
910 (debug)          { NS }
911 (debug)          { g__tag_struct_tag_NS_tl }

912          %\prop_show:c { g__tag_struct_g__tag_struct_stack_current_tl _prop }
913          %\seq_show:c {g__tag_struct_kids_l__tag_struct_stack_parent_tmpa_tl _seq}
914 (debug) \__tag_debug_struct_begin_insert:n { #1 }
915          \group_end:
916      }
917 (debug){ \__tag_debug_struct_begin_ignore:n { #1 }}
918      }
919 (package)\cs_set_protected:Nn \tag_struct_end:
920 (debug)\cs_set_protected:Nn \tag_struct_end:
921      { %take the current structure num from the stack:
922          %the objects are written later, lua mode hasn't all needed info yet
923          %\seq_show:N \g__tag_struct_stack_seq
924 (package)\__tag_check_if_active_struct:T
925 (debug)\__tag_check_if_active_struct:TF
926          {
927          \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
928          \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
929          {
930          \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
931          }
932          { \__tag_check_no_open_struct: }
933          % get the previous one, shouldn't be empty as the root should be there
934          \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
935          {
936          \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
937          }
938          {
939          \__tag_check_no_open_struct:
940          }
941          \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
942          {
943          \tl_gset:Ne \g__tag_struct_tag_tl

```

```

944         { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
945         \prop_get:NVNT\g__tag_role_tags_NS_prop \g__tag_struct_tag_tl\l__tag_tmpa_tl
946         {
947             \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
948         }
949     }
950 <debug>\__tag_debug_struct_end_insert:
951 }
952 <debug>{\__tag_debug_struct_end_ignore:}
953 }
954
955 \cs_set_protected:Npn \tag_struct_end:n #1
956 {
957 <debug>    \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
958     \tag_struct_end:
959 }
960 </package | debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 98.)

\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.

```

961 <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
962 <*package | debug>
963 \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
964 {
965     \__tag_check_if_active_struct:T
966     {
967         \prop_if_exist:cTF
968         { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop }
969         {
970             \__tag_check_struct_used:n {#1}
971             %add the label structure as kid to the current structure (can be the root)
972             \__tag_struct_kid_struct_gput_right:ee
973             { \g__tag_struct_stack_current_tl }
974             { \__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0} }
975             %add the current structure to the labeled one as parents
976             \__tag_prop_gput:cne
977             { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
978             { P }
979             {
980                 \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
981             }
982         }
983     }
984 }
985
986 debug code
987 <debug>    \prop_gput:cne
988 <debug>    { g__tag_struct_debug_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{
989 <debug>    { P }
990 <debug>    {
991 <debug>        parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
992 <debug>        \g__tag_struct_tag_tl
993 <debug>    }

```

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

```

989         \__tag_struct_get_parentrole:eNN
990         {\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0}}
991         \l__tag_tmpa_tl
992         \l__tag_tmpb_tl
993     \__tag_check_parent_child:VVVVN
994         \g__tag_struct_tag_tl
995         \g__tag_struct_tag_NS_tl
996         \l__tag_tmpa_tl
997         \l__tag_tmpb_tl
998         \l__tag_parent_child_check_tl
999     \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1000     {
1001         \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1002         \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1003         \__tag_role_remap:
1004         \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1005         \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1006         \__tag_struct_set_tag_info:eVV
1007         { \int_use:N \c@g__tag_struct_abs_int }
1008         \g__tag_struct_tag_tl
1009         \g__tag_struct_tag_NS_tl
1010     }
1011 }
1012 {
1013     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1014 }
1015 }
1016 }
1017 </package | debug>

```

(End of definition for `\tag_struct_use:n`. This function is documented on page 98.)

`\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.

```

1018 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1019 <*package | debug>
1020 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1021 {
1022     \__tag_check_if_active_struct:T
1023     {
1024         \prop_if_exist:cTF
1025         { g__tag_struct_#1_prop } %
1026         {
1027             \prop_get:cnNT
1028             {g__tag_struct_#1_prop}
1029             {P}
1030             \l__tag_tmpa_tl
1031             {
1032                 \msg_warning:nnn { tag } {struct-used-twice} {#1}
1033             }
1034             %add the \#1 structure as kid to the current structure (can be the root)

```

```

1035     \__tag_struct_kid_struct_gput_right:ee
1036     { \g__tag_struct_stack_current_tl }
1037     { #1 }
1038     %add the current structure to \#1 as parent
1039     \__tag_struct_prop_gput:nne
1040     { #1 }
1041     { P }
1042     {
1043     \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
1044     }
1045     (debug)     \prop_gput:cne
1046     (debug)     { g__tag_struct_debug_#1_prop }
1047     (debug)     { P }
1048     (debug)     {
1049     (debug)     parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1050     (debug)     \g__tag_struct_tag_tl
1051     (debug)     }

```

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

```

1052     \__tag_struct_get_parentrole:eNN
1053     {#1}
1054     \l__tag_tmpa_tl
1055     \l__tag_tmpb_tl
1056     \__tag_check_parent_child:VVVVN
1057     \g__tag_struct_tag_tl
1058     \g__tag_struct_tag_NS_tl
1059     \l__tag_tmpa_tl
1060     \l__tag_tmpb_tl
1061     \l__tag_parent_child_check_tl
1062     \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1063     {
1064     \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1065     \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1066     \__tag_role_remap:
1067     \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1068     \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1069     \__tag_struct_set_tag_info:eVV
1070     { \int_use:N \c@g__tag_struct_abs_int }
1071     \g__tag_struct_tag_tl
1072     \g__tag_struct_tag_NS_tl
1073     }
1074     }
1075     {
1076     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1077     }
1078     }
1079     }
1080 </package | debug>

```

(End of definition for \tag_struct_use_num:n. This function is documented on page 98.)

\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.

```

1081 (*package)
1082 \cs_new:Npn \tag_struct_object_ref:n #1
1083 {
1084   \pdf_object_ref:n {__tag/struct/#1}
1085 }
1086 \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 98.)

\tag_struct_gput:nnn This is a command that allows to update the data of a structure. This often can't be done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the only keyword is `ref` which updates the `Ref` key (an array)

```

1087 \cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1088 {
1089   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1090   { %warning??
1091     \use_none:nn
1092   }
1093   {#1}{#3}
1094 }
1095 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1096 </package>

```

(End of definition for \tag_struct_gput:nnn. This function is documented on page 99.)

__tag_struct_gput_data_ref:nn

```

1097 (*package)
1098 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1099 % #1 receiving struct num, #2 list of object ref
1100 {
1101   \prop_get:cnN
1102   { g__tag_struct_#1_prop }
1103   {Ref}
1104   \l__tag_get_tmpc_tl
1105   \__tag_struct_prop_gput:nne
1106   { #1 }
1107   { Ref }
1108   { \quark_if_no_value:NF\l__tag_get_tmpc_tl { \l__tag_get_tmpc_tl\c_space_tl }#2 }
1109 }
1110 \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 **\tag_struct_insert_annot:ee** **\tag_struct_insert_annot:ee** These are the user commands to insert annotations. They must be used together to get the numbers right. They use a counter to the `StructParent` and `\tag_struct_insert_annot:nn` increases the counter given back by `\tag_struct_parent_int:`.

\tag_struct_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.

```

1111 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1112 % #2 struct parent num
1113 {
1114   \__tag_check_if_active_struct:T

```

```

1115     {
1116     \tag_struct_insert_annot:nn {#1}{#2}
1117     }
1118   }
1119
1120 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1121 \cs_new:Npn \tag_struct_parent_int: {\int_use:c {c@g__tag_parenttree_obj_int }}
1122
1123 \end{package}
1124

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:`. These functions are documented on page 98.)

7 Attributes and attribute classes

```

1125 \begin{header}
1126 \ProvidesExplPackage {tagpdf-attr-code} {2024-02-29} {0.98x}
1127 {part of tagpdf - code related to attributes and attribute classes}
1128 \end{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.
`\g__tag_attr_objref_prop` `\l__@@_attr_value_tl` is used to build the attribute array or key. Every time an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in `\g__@@_attr_objref_prop`

```

1129 \begin{package}
1130 \prop_new:N \g__tag_attr_entries_prop
1131 \seq_new:N \g__tag_attr_class_used_seq
1132 \tl_new:N \l__tag_attr_value_tl
1133 \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.
`role/new-attribute_␣(setup-key)` `role/new-attribute` expects two brace group, the name and the content. The content typically needs an `/O` key for the owner. An example look like this.

TODO: consider to put them directly in the ClassMap, that is perhaps more effective.

```

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

```

```

1134 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1135 {
1136   \prop_gput:Nen \g__tag_attr_entries_prop
1137   {\pdf_name_from_unicode_e:n{#1}}{#2}
1138 }
1139
1140 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1141 \keys_define:nn { __tag / setup }
1142 {
1143   role/new-attribute .code:n =
1144   {
1145     \__tag_attr_new_entry:nn #1
1146   }

```

deprecated name

```

1147   ,newattribute .code:n =
1148   {
1149     \__tag_attr_new_entry:nn #1
1150   },
1151 }

```

(End of definition for `__tag_attr_new_entry:nn`, `role/new-attribute (setup-key)`, and `newattribute (deprecated)`. These functions are documented on page 101.)

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

```

1152 \keys_define:nn { __tag / struct }
1153 {
1154   attribute-class .code:n =
1155   {
1156     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1157     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist

```

we convert the names into pdf names with slash

```

1158     \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1159     {
1160       \pdf_name_from_unicode_e:n {##1}
1161     }
1162     \seq_map_inline:Nn \l__tag_tmpa_seq
1163     {
1164       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1165       {
1166         \msg_error:nnn { tag } { attr-unknown } { ##1 }
1167       }
1168       \seq_gput_left:Nn\g__tag_attr_class_used_seq { ##1}
1169     }
1170     \tl_set:Ne \l__tag_tmpa_tl
1171     {
1172       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1173       \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1174       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1175     }
1176     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1177     {
1178       \__tag_struct_prop_gput:nne

```

```

1179         { \int_use:N \c@g__tag_struct_abs_int }
1180         { C }
1181         { \l__tag_tmpa_tl }
1182         %\prop_show:c { g__tag_struct \int_eval:n {\c@g__tag_struct_abs_int}_prop }
1183     }
1184 }
1185 }

```

(End of definition for attribute-class (struct-key). This function is documented on page 101.)

attribute_(struct-key)

```

1186 \keys_define:nn { __tag / struct }
1187 {
1188     attribute .code:n = % A property (attribute, value currently a dictionary)
1189     {
1190         \clist_set:Ne          \l__tag_tmpa_clist { #1 }
1191         \clist_if_empty:NF \l__tag_tmpa_clist
1192         {
1193             \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1194             we convert the names into pdf names with slash
1195             \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1196             {
1197                 \pdf_name_from_unicode_e:n {##1}
1198             }
1199             \tl_set:Ne \l__tag_attr_value_tl
1200             {
1201                 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}%]
1202             }
1203             \seq_map_inline:Nn \l__tag_tmpa_seq
1204             {
1205                 \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1206                 {
1207                     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1208                 }
1209                 \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
1210                 {%\prop_show:N \g__tag_attr_entries_prop
1211                 \pdf_object_unnamed_write:ne
1212                 { dict }
1213                 {
1214                     \prop_item:Nn \g__tag_attr_entries_prop {##1}
1215                 }
1216                 \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1217             }
1218             \tl_put_right:Ne \l__tag_attr_value_tl
1219             {
1220                 \c_space_tl
1221                 \prop_item:Nn \g__tag_attr_objref_prop {##1}
1222             }
1223             % \tl_show:N \l__tag_attr_value_tl
1224             }
1225             \tl_put_right:Ne \l__tag_attr_value_tl
1226             { %[
1227                 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}%]
1228             }
1229         }
1230     }

```

```

1228     % \tl_show:N \l__tag_attr_value_tl
1229     \__tag_struct_prop_gput:nne
1230     { \int_use:N \c@g__tag_struct_abs_int }
1231     { A }
1232     { \l__tag_attr_value_tl }
1233   }
1234 },
1235 }
1236 \end{package}

```

(End of definition for attribute (struct-key). This function is documented on page 101.)

Part VIII

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2024-02-29} {0.98x}
4   {tagpdf-driver-for-luatex}
```

1 Loading the lua

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

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

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

```

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

```

30 {
31   \prop_gput:Nnn #1 { #2 } { #3 }
32   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 ["#2"] = "#3" }
33 }
34
35
36 \cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
37 {
38   \seq_gput_right:Nn #1 { #2 }
39   \lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }
40 }
41
42 %Hm not quite sure about the naming
43
44 \cs_set:Npn \__tag_seq_item:cn #1 #2
45 {
46   \lua_now:e { tex.print(ltx.__tag.tables.#1[#2]) }
47 }
48
49 \cs_set:Npn \__tag_prop_item:cn #1 #2
50 {
51   \lua_now:e { tex.print(ltx.__tag.tables.#1["#2"]) }
52 }
53
54 %for debugging commands that show both the seq/prop and the lua tables
55 \cs_set_protected:Npn \__tag_seq_show:N #1
56 {
57   \seq_show:N #1
58   \lua_now:e { ltx.__tag.trace.log ("lua-sequence-array~\cs_to_str:N#1",1) }
59   \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables.\cs_to_str:N#1) }
60 }
61
62 \cs_set_protected:Npn \__tag_prop_show:N #1
63 {
64   \prop_show:N #1
65   \lua_now:e {ltx.__tag.trace.log ("lua-property-table~\cs_to_str:N#1",1) }
66   \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables.\cs_to_str:N#1) }
67 }

```

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

68 </luatex>

The module declaration

```

69 (*lua)
70 -- tagpdf.lua
71 -- Ulrike Fischer
72
73 local ProvidesLuaModule = {
74   name           = "tagpdf",
75   version        = "0.98x",      --TAGVERSION
76   date           = "2024-02-29", --TAGDATE
77   description    = "tagpdf lua code",
78   license        = "The LATEX Project Public License 1.3c"
79 }

```

```

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

```

Some comments about the lua structure.

```

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

```

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

The `interwordspace attr` is set by the function `@@_mark_spaces`, and marks the place where spaces should be inserted. The `interwordfont attr` is set by the function `@@_mark_spaces` too and stores the font, so that we can decide which font to use for the real space char.

```

134 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
135 local mcntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
136 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
137 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

```

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

```

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

```

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

```

171 ltx          = ltx          or { }

```

```

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

```

183 local __tag_log =
184 function (message,loglevel)
185   if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
186     texio.write_nl("tagpdf: ".. message)
187   end
188 end
189
190 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.

```

191 function ltx.__tag.trace.show_seq (seq)
192   if (type(seq) == "table") then
193     for i,v in ipairs(seq) do
194       __tag_log ("[" .. i .. "] => " .. tostring(v),1)
195     end
196   else
197     __tag_log ("sequence " .. tostring(seq) .. " not found",1)
198   end
199 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.

`ltx.__tag.trace.show_prop`

```

200 local __tag_pairs_prop =
201 function (prop)
202   local a = {}
203   for n in pairs(prop) do tableinsert(a, n) end
204   table.sort(a)
205   local i = 0 -- iterator variable
206   local iter = function () -- iterator function
207     i = i + 1

```

```

208         if a[i] == nil then return nil
209         else return a[i], prop[a[i]]
210         end
211     end
212     return iter
213 end
214
215
216 function ltx.__tag.trace.show_prop (prop)
217 if (type(prop) == "table") then
218     for i,v in __tag_pairs_prop (prop) do
219         __tag_log ("[" .. i .. "] => " .. tostring(v),1)
220     end
221 else
222     __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
223 end
224 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

```

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

```

240 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
241 for i = min, max do
242     ltx.__tag.trace.show_mc_data (i,loglevel)
243 end
244 texio.write_nl("")
245 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.

```

246 function ltx.__tag.trace.show_struct_data (num)
247 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
248     for k,v in ipairs(ltx.__tag.struct[num]) do

```

```

249   __tag_log ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
250   end
251   else
252     __tag_log ("struct "..num.." not found ",1)
253   end
254 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.

```

255 local __tag_get_mc_cnt_type_tag = function (n)
256   local mccnt      = nodegetattribute(n,mccntattributeid) or -1
257   local mctype     = nodegetattribute(n,mctypeattributeid) or -1
258   local tag        = ltx.__tag.func.get_tag_from(mctype)
259   return mccnt,mctype,tag
260 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.

```

261 local function __tag_get_mathsubtype (mathnode)
262   if mathnode.subtype == 0 then
263     subtype = "beginmath"
264   else
265     subtype = "endmath"
266   end
267   return subtype
268 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.

```

269 ltx.__tag.tables.role_tag_attribute = {}
270 ltx.__tag.tables.role_attribute_tag = {}

```

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

`ltx.__tag.func.alloctag`

```

271 local __tag_alloctag =
272   function (tag)
273     if not ltx.__tag.tables.role_tag_attribute[tag] then
274       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
275       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
276       __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
277     end
278   end
279 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
`ltx.__tag.func.get_num_from` it recorded (and so the attribute value). The first function outputs the number for lua,
`ltx.__tag.func.output_num_from` while the `output` function outputs to tex.

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

```
299 local __tag_get_tag_from =
300 function (num)
301   if ltx.__tag.tables.role_attribute_tag[num] then
302     a = ltx.__tag.tables.role_attribute_tag[num]
303   else
304     a= "UNKNOWN"
305   end
306   return a
307 end
308
309 ltx.__tag.func.get_tag_from = __tag_get_tag_from
310
311 function ltx.__tag.func.output_tag_from (num)
312   tex.sprint(catlatex,__tag_get_tag_from (num))
313 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.

```
314 function ltx.__tag.func.store_mc_data (num,key,data)
315   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
```

```

316 ltx.__tag.mc[num][key] = data
317 __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
318 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.

```

319 function ltx.__tag.func.store_mc_label (label,num)
320 ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
321 ltx.__tag.mc.labels[label] = num
322 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.

```

323 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
324 ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
325 ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
326 local kidtable = {kid=kid,page=page}
327 tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
328 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.

```

329 function ltx.__tag.func.mc_num_of_kids (mcnum)
330 local num = 0
331 if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
332   num = #ltx.__tag.mc[mcnum]["kids"]
333 end
334 ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
335 return num
336 end

```

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

3.2 Functions to insert the pdf literals

__tag_backend_create_emc_node This insert the emc node. We support also dvips and dvipdfmx backend

```

__tag_insert_emc_node
337 local __tag_backend_create_emc_node
338 if tex.outputmode == 0 then
339   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
340     function __tag_backend_create_emc_node ()
341       local emcnode = nodenew("whatsit","special")
342       emcnode.data = "pdf:code EMC"
343       return emcnode
344     end
345   else -- assume a dvips variant
346     function __tag_backend_create_emc_node ()
347       local emcnode = nodenew("whatsit","special")
348       emcnode.data = "ps:SDict begin mark /EMC pdfmark end"

```

```

349     return emcnode
350   end
351 end
352 else -- pdf mode
353   function __tag_backend_create_emc_node ()
354     local emcnode = nodenew("whatsit","pdf_literal")
355     emcnode.data = "EMC"
356     emcnode.mode=1
357     return emcnode
358   end
359 end
360
361 local function __tag_insert_emc_node (head,current)
362   local emcnode= __tag_backend_create_emc_node()
363   head = node.insert_before(head,current,emcnode)
364   return head
365 end

```

(End of definition for __tag_backend_create_emc_node and __tag_insert_emc_node.)

```

__tag_backend_create_bmc_node This inserts a simple bmc node
__tag_insert_bmc_node
366 local __tag_backend_create_bmc_node
367 if tex.outputmode == 0 then
368   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
369     function __tag_backend_create_bmc_node (tag)
370       local bmcnode = nodenew("whatsit","special")
371       bmcnode.data = "pdf:code /"..tag.." BMC"
372       return bmcnode
373     end
374   else -- assume a dvips variant
375     function __tag_backend_create_bmc_node (tag)
376       local bmcnode = nodenew("whatsit","special")
377       bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
378       return bmcnode
379     end
380   end
381 else -- pdf mode
382   function __tag_backend_create_bmc_node (tag)
383     local bmcnode = nodenew("whatsit","pdf_literal")
384     bmcnode.data = "/"..tag.." BMC"
385     bmcnode.mode=1
386     return bmcnode
387   end
388 end
389
390 local function __tag_insert_bmc_node (head,current,tag)
391   local bmcnode = __tag_backend_create_bmc_node (tag)
392   head = node.insert_before(head,current,bmcnode)
393   return head
394 end

```

(End of definition for __tag_backend_create_bmc_node and __tag_insert_bmc_node.)

```

__tag_backend_create_bdc_node This inserts a bcd node with a fix dict. TODO: check if this is still used, now that we
__tag_insert_bdc_node create properties.

```

```

395 local __tag_backend_create_bdc_node
396
397 if tex.outputmode == 0 then
398   if token.get_macro("c_sys_backend_str") == "dviPDFmx" then
399     function __tag_backend_create_bdc_node (tag,dict)
400       local bdcnode = nodenew("whatsit","special")
401       bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
402       return bdcnode
403     end
404   else -- assume a dvips variant
405     function __tag_backend_create_bdc_node (tag,dict)
406       local bdcnode = nodenew("whatsit","special")
407       bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
408       return bdcnode
409     end
410   end
411 else -- pdf mode
412   function __tag_backend_create_bdc_node (tag,dict)
413     local bdcnode = nodenew("whatsit","pdf_literal")
414     bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
415     bdcnode.mode=1
416     return bdcnode
417   end
418 end
419
420 local function __tag_insert_bdc_node (head,current,tag,dict)
421   bdcnode= __tag_backend_create_bdc_node (tag,dict)
422   head = node.insert_before(head,current,bdcnode)
423   return head
424 end

```

(End of definition for __tag_backend_create_bdc_node and __tag_insert_bdc_node.)

`__tag_pdf_object_ref` This allows to reference a pdf object reserved with the `l3pdf` command by name. The return value is `n 0 R`, if the object doesn't exist, `n` is 0. TODO: it uses internal `l3pdf` commands, this should be properly supported by `l3pdf`

```

425 local function __tag_pdf_object_ref (name)
426   local tokename = 'c_pdf_backend_object_'..name..'_int'
427   local object = token.create(tokename).mode .. ' 0 R'
428   return object
429 end
430 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 insert red color markers in the places where space chars can go, it can have side effects so not always reliable, but ok.

```

431 local function __tag_show_spacemark (head,current,color,height)
432   local markcolor = color or "1 0 0"
433   local markheight = height or 10
434   local pdfstring

```

```

435 if tex.outputmode == 0 then
436   -- ignore dvi mode for now
437 else
438   pdfstring = node.new("whatsit","pdf_literal")
439   pdfstring.data =
440     string.format("q ".markcolor.." RG ".markcolor.." rg 0.4 w 0 %g m 0 %g 1 S Q",-
3,markheight)
441   head = node.insert_after(head,current,pdfstring)
442   return head
443 end
444 end

```

(End of definition for `__tag_show_spacemark`.)

`__tag_fakespace` This is used to define a lua version of `\pdf-fakespace`

```

ltx.__tag.func.fakespace 445 local function __tag_fakespace()
446   tex.setattribute(iwspaceattributeid,1)
447   tex.setattribute(iwfontattributeid,font.current())
448 end
449 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.

```

450 --[[ a function to mark up places where real space chars should be inserted
451     it only sets an attribute.
452 --]]
453
454 local function __tag_mark_spaces (head)
455   local inside_math = false
456   for n in nodetraverse(head) do
457     local id = n.id
458     if id == GLYPH then
459       local glyph = n
460       default_currfontid = glyph.font
461       if glyph.next and (glyph.next.id == GLUE)
462         and not inside_math and (glyph.next.width >0)
463       then
464         nodesetattribute(glyph.next,iwspaceattributeid,1)
465         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
466         -- for debugging
467         if ltx.__tag.trace.showspace then
468           __tag_show_spacemark (head,glyph)
469         end
470       elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
471         local kern = glyph.next
472         if kern.next and (kern.next.id== GLUE)  and (kern.next.width >0)
473       then
474         nodesetattribute(kern.next,iwspaceattributeid,1)
475         nodesetattribute(kern.next,iwfontattributeid,glyph.font)
476       end
477     end

```

```

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

```
528 local default_space_char = nodenew(GLYPH)
529 local default_fontid     = fontid("TU/lmr/m/n/10")
530 local default_currfontid = fontid("TU/lmr/m/n/10")
531 default_space_char.char  = 32
532 default_space_char.font  = default_fontid
```

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

```
533 local function __tag_font_has_space (fontid)
534   t= fonts.hashes.identifiers[fontid]
535   if luaotfload.aux.slot_of_name(fontid,"space")
536     or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
537   then
538     return true
539   else
540     return false
541   end
542 end
```

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

```
__tag_space_chars_shipout
ltx.__tag.func.space_chars_shipout
543 local function __tag_space_chars_shipout (box)
544   local head = box.head
545   if head then
546     for n in node.traverse(head) do
547       local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
548       if n.id == HLIST then -- enter the hlist
549         __tag_space_chars_shipout (n)
550       elseif n.id == VLIST then -- enter the vlist
551         __tag_space_chars_shipout (n)
552       elseif n.id == GLUE then
553         if ltx.__tag.trace.showspace and spaceattr==1 then
554           __tag_show_spacemark (head,n,"0 1 0")
555         end
556         if spaceattr==1 then
557           local space
558           local space_char = node.copy(default_space_char)
559           local curfont     = nodegetattribute(n,iwfontattributeid)
560           ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
561           if curfont and
562             -- luaotfload.aux.slot_of_name(curfont,"space")
563             __tag_font_has_space (curfont)
564           then
565             space_char.font=curfont
566           end
567           head, space = node.insert_before(head, n, space_char) --
568           n.width     = n.width - space.width
569           space.attr  = n.attr
570         end
571       end
572     end
573     box.head = head
```

```

574 end
575 end
576
577 function ltx.__tag.func.space_chars_shipout (box)
578   __tag_space_chars_shipout (box)
579 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.

```

580 function ltx.__tag.func.mc_insert_kids (mcnum, single)
581   if ltx.__tag.mc[mcnum] then
582     ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
583     if ltx.__tag.mc[mcnum]["kids"] then
584       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
585         tex.sprint("[")
586       end
587       for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
588         local kidnum = kidstable["kid"]
589         local kidpage = kidstable["page"]
590         local kidpageobjnum = pdfpageref(kidpage)
591         ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
592           " insert KID " .. i ..
593           " with num " .. kidnum ..
594           " on page " .. kidpage.."/"..kidpageobjnum,3)
595         tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> " )
596       end
597       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
598         tex.sprint("]")
599       end
600     else
601       -- this is typically not a problem, e.g. empty hbox in footer/header can
602       -- trigger this warning.
603       ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
604       if single==1 then
605         tex.sprint("null")
606       end
607     end
608   else
609     ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
610   end
611 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.

```

612 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)

```

```

613 ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
614 ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
615 -- a structure can contain more than on mc chunk, the content should be ordered
616 tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
617 ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
618                 mcnum.." inserted in struct "..structnum,3)
619 -- but every mc can only be in one structure
620 ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
621 ltx.__tag.mc[mcnum]["parent"] = structnum
622 end
623

```

(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.

```

624 -- pay attention: lua counts arrays from 1, tex pages from one
625 -- mcid and arrays in pdf count from 0.
626 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagencnt,page)
627 ltx.__tag.page[page] = ltx.__tag.page[page] or {}
628 ltx.__tag.page[page][mcpagencnt] = mcnum
629 ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
630                 ": inserting MCID " .. mcpagencnt .. " => " .. mcnum,3)
631 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)

```

632 local function __tag_update_mc_attributes (head,mcnum,type)
633 for n in node.traverse(head) do
634     node.set_attribute(n,mccntattributeid,mcnum)
635     node.set_attribute(n,mctypeattributeid,type)
636     if n.id == HLIST or n.id == VLIST then
637         __tag_update_mc_attributes (n.list,mcnum,type)
638     end
639 end
640 return head
641 end
642 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.

```

643 --[[
644     Now follows the core function
645     It wades through the shipout box and checks the attributes
646     ARGUMENTS
647     box: is a box,
648     mcpagencnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
649     mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a c
650     mcopen: num, records if some bdc/emc is open
651     These arguments are only needed for log messages, if not present are replaces by fix stri

```

```

652     name: string to describe the box
653     mctypeprev: num, the type attribute of the previous node/whatever
654
655     there are lots of logging messages currently. Should be cleaned up in due course.
656     One should also find ways to make the function shorter.
657 --]]
658
659 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mcntprev,mcopen,name,mctypeprev)
660     local name = name or ("SOMEBOX")
661     local mctypeprev = mctypeprev or -1
662     local abspage = status.total_pages + 1 -- the real counter is increased
663                                           -- inside the box so one off
664                                           -- if the callback is not used. (???)
665     ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
666     ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
667                         " prev "..mcntprev ..
668                         " type prev "..mctypeprev,4)
669     ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
670                         " TYPE ".. node.type(node.getid(box)),3)
671     local head = box.head -- ShipoutBox is a vlist?
672     if head then
673         mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
674         ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
675                             node.type(node.getid(head))..
676                             " MC"..tostring(mccnthead)..
677                             " => TAG " .. tostring(mctypehead)..
678                             " => ".. tostring(taghead),3)
679     else
680         ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
681                             tostring(head),3)
682     end
683     for n in node.traverse(head) do
684         local mcnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
685         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
686         ltx.__tag.trace.log ("INFO TAG-NODE: "..
687                             node.type(node.getid(n))..
688                             " MC"..tostring(mcnt)..
689                             " => TAG ".. tostring(mctype)..
690                             " => " .. tostring(tag),3)
691         if n.id == HLIST
692         then -- enter the hlist
693             mcopen,mcpagecnt,mcntprev,mctypeprev=
694             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcntprev,mcopen,"INTERNAL HLIST",mctypeprev)
695         elseif n.id == VLIST then -- enter the vlist
696             mcopen,mcpagecnt,mcntprev,mctypeprev=
697             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcntprev,mcopen,"INTERNAL VLIST",mctypeprev)
698         elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but th
699                                                     -- been done if the previous shipout wandering, so here it
700         elseif n.id == LOCAL_PAR then -- local_par is ignored
701         elseif n.id == PENALTY then -- penalty is ignored
702         elseif n.id == KERN then -- kern is ignored
703             ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
704                                 node.type(node.getid(n)).." ".n.subtype,4)
705         else

```

```

706 -- math is currently only logged.
707 -- we could mark the whole as math
708 -- for inner processing the mlist_to_hlist callback is probably needed.
709 if n.id == MATH then
710   ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
711     node.type(node.getid(n)).." " ..__tag_get_mathsubtype(n),4)
712 end
713 -- endmath
714 ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
715   mccnt.." prev "..mccntprev,4)
716 if mccnt~=mccntprev then -- a new mc chunk
717   ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
718     node.type(node.getid(n))..
719     " MC"..tostring(mccnt)..
720     " <=> PREVIOUS "..tostring(mccntprev),4)
721 if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
722   box.list=__tag_insert_emc_node (box.list,n)
723   mcopen = mcopen - 1
724   ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
725     mcpagecnt .. " MCOOPEN = " .. mcopen,3)
726   if mcopen ~=0 then
727     ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,1)
728   end
729 end
730 if ltx.__tag.mc[mccnt] then
731   if ltx.__tag.mc[mccnt]["artifact"] then
732     ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
733       tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
734     if ltx.__tag.mc[mccnt]["artifact"] == "" then
735       box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
736     else
737       box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]
738     end
739   else
740     ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
741       tostring(tag),3)
742     mcpagecnt = mcpagecnt +1
743     ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
744     local dict= "/MCID "..mcpagecnt
745     if ltx.__tag.mc[mccnt]["raw"] then
746       ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
747         tostring(ltx.__tag.mc[mccnt]["raw"]),3)
748       dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
749     end
750     if ltx.__tag.mc[mccnt]["alt"] then
751       ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
752         tostring(ltx.__tag.mc[mccnt]["alt"]),3)
753       dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
754     end
755     if ltx.__tag.mc[mccnt]["actualtext"] then
756       ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
757         tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
758       dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
759     end

```

```

760     box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
761     ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
762     ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
763     ltx.__tag.trace.show_mc_data (mccnt,3)
764     end
765     mcopen = mcopen + 1
766     else
767     if tagunmarkedbool.mode == truebool.mode then
768     ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
769     box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
770     mcopen = mcopen + 1
771     else
772     ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
773     end
774     end
775     mccntprev = mccnt
776     end
777     end -- end if
778 end -- end for
779 if head then
780     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
781     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
782         node.type(node.getid(head))..
783         " MC"..tostring(mccnthead)..
784         " => TAG "..tostring(mctypehead)..
785         " => "..tostring(taghead),4)
786     else
787     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
788     end
789     ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
790         tostring(name)..
791         " TYPE ".. node.type(node.getid(box)),4)
792     return mcopen,mcpagecnt,mccntprev,mctypeprev
793 end
794

```

(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.

```

795 function ltx.__tag.func.mark_shipout (box)
796     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
797     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
798     local emcnode = __tag_backend_create_emc_node ()
799     local list = box.list
800     if list then
801     list = node.insert_after (list,node.tail(list),emcnode)
802     mcopen = mcopen - 1
803     ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEEN " .. mcopen,3)
804     else
805     ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
806     end
807     if mcopen ~=0 then

```

```

808     ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
809   end
810 end
811 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.

```

812 function ltx.__tag.func.fill_parent_tree_line (page)
813   -- we need to get page-> i=kid -> mcnum -> structnum
814   -- pay attention: the kid numbers and the page number in the parent tree start with 0!
815   local numsentry = ""
816   local pdfpage = page-1
817   if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
818     mcchunks=#ltx.__tag.page[page]
819     ltx.__tag.trace.log("INFO PARENTTREE-NUM: page "..
820       page.." has "..mcchunks.." +1 Elements ",4)
821     for i=0,mcchunks do
822       -- what does this log??
823       ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: "..
824         ltx.__tag.page[page][i],4)
825     end
826     if mcchunks == 0 then
827       -- only one chunk so no need for an array
828       local mcnum = ltx.__tag.page[page][0]
829       local structnum = ltx.__tag.mc[mcnum]["parent"]
830       local propname = "g__tag_struct_"..structnum.."_prop"
831       --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
832       local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
833       ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF: =====>"..
834         tostring(objref),5)
835       numsentry = pdfpage .. " [".. objref .. "]"
836       ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
837         page.. " num entry = ".. numsentry,3)
838     else
839       numsentry = pdfpage .. " ["
840       for i=0,mcchunks do
841         local mcnum = ltx.__tag.page[page][i]
842         local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
843         local propname = "g__tag_struct_"..structnum.."_prop"
844         --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
845         local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
846         numsentry = numsentry .. " " .. objref
847       end
848       numsentry = numsentry .. "]"
849       ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
850         page.. " num entry = ".. numsentry,3)
851     end
852   else
853     ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)

```

```

854     numsentry = pdfpage.." []"
855     end
856     return numsentry
857 end
858
859 function ltx.__tag.func.output_parenttree (abspage)
860   for i=1,abspage do
861     line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
862     tex.sprint(catlatex,line)
863   end
864 end

```

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

```

865 </lua>

```

Part IX

The tagpdf-roles module

Tags, roles and namespace code

Part of the tagpdf package

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

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

The key-value list knows the following keys:

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

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

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

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

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

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

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

```
1 <@<tag>
2 <*/header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2024-02-29} {0.98x}
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/pdf/ssn>

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

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

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

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

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

`9 \prop_new:N \g__tag_role_NS_prop`

(End of definition for \g__tag_role_NS_prop.)

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

`10 \prop_new:N \g__tag_role_index_prop`

(End of definition for \g__tag_role_index_prop.)

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

`11 \prop_new:N \l__tag_role_debug_prop`

(End of definition for \l__tag_role_debug_prop.)

We need also a bunch of temporary variables.

`\l__tag_role_tag_tmpa_tl`

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

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

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

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

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

`17 \seq_new:N \l__tag_role_tmpa_seq`

(End of definition for \l__tag_role_tag_tmpa_tl and others.)

1.2 Namespaces

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

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

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

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

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

```
\__tag_role_NS_new:nnn
```

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

```

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

```

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

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

`\c__tag_role_userNS_id_str`

```

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

```

(End of definition for `\c__tag_role_userNS_id_str`.)

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

```

73 \bool_new:N \g__tag_role_add_mathml_bool
74 \__tag_role_NS_new:nnn {pdf} {http://iso.org/pdf/ssn}{}
75 \__tag_role_NS_new:nnn {pdf2} {http://iso.org/pdf2/ssn}{}
76 \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{}
77 \__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dfl1t/2022}{}
78 \__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book/2022}{}
79 \exp_args:Nne
80   \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{}

```

1.3 Adding a new tag

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

`__tag_role_alloctag:nnm`

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

```

81 \pdf_version_compare:NnTF < {2.0}
82   {
83     \sys_if_engine luatex:TF
84     {

```

```

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

```

(End of definition for __tag_role_alloctag:nnn.)

1.3.1 pdf 1.7 and earlier

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

```

127 \cs_new_protected:Nn \__tag_role_add_tag:nn % (new) name, reference to old
128 {
checks and messages

```

```

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

```

now the addition

```

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

```

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

```

143 \tl_if_empty:nF { #2 }
144 {
145   \prop_get:NnN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
146   \quark_if_no_value:NTF \l__tag_tmpa_tl
147   {
148     \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
149   }
150   {
151     \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
152   }
153 }
154 }
155 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV,ne}

```

(End of definition for __tag_role_add_tag:nn.)

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

__tag_role_get:nnNN

```

156 \pdf_version_compare:NnT < {2.0}
157 {
158   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 %#1 tag, #2 NS, #3 tlvar which hold the role tag
159   {
160     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
161     {
162       \tl_set:Nn #3 {#1}
163     }
164     \tl_set:Nn #4 {}
165   }
166   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
167 }
168

```

(End of definition for __tag_role_get:nnNN.)

1.3.2 The pdf 2.0 version

```
\__tag_role_add_tag:nmmm The pdf 2.0 version takes four arguments: tag/namespace/role/namespace
169 \cs_new_protected:Nn \__tag_role_add_tag:nmmm %tag/namespace/role/namespace
170 {
171   \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
172   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
173   {
174     \msg_info:nnn { tag }{new-tag}{#1}
175   }
176   \prop_if_exist:cTF
177   { g__tag_role_NS_#4_class_prop }
178   {
179     \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
180     \quark_if_no_value:NT \l__tag_tmpa_tl
181     {
182       \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
183     }
184   }
185   { \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--} }
186   \__tag_role_alloctag:nnV {#1}{#2}\l__tag_tmpa_tl
```

Do not remap standard tags. TODO add warning?

```
187   \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
188   {
189     \pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
190     {
191       [
192         \pdf_name_from_unicode_e:n{#3}
193         \c_space_tl
194         \pdf_object_ref:n {tag/NS/#4}
195       ]
196     }
197   }
```

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

```
198   \tl_if_empty:nF { #2 }
199   {
200     \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
201     \quark_if_no_value:NTF \l__tag_tmpa_tl
202     {
203       \prop_gput:cne { g__tag_role_NS_#2_prop } {#1}
204       {{\tl_to_str:n{#3}}{\tl_to_str:n{#4}}}
205     }
206     {
207       \prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
208     }
209   }
```

We also store into the pdf 1.7 rolemapping so that we can add that as fallback for pdf 1.7 processor

```
210   \bool_if:NT \l__tag_role_update_bool
211   {
212     \tl_if_empty:nF { #3 }
```

```

213     {
214       \tl_if_eq:nnF{#1}{#3}
215       {
216         \prop_get:Nn \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
217         \quark_if_no_value:NTF \l__tag_tmpa_tl
218         {
219           \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
220         }
221         {
222           \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
223         }
224       }
225     }
226   }
227 }
228 \cs_generate_variant:Nn \__tag_role_add_tag:nmmm {VVVV}

```

(End of definition for __tag_role_add_tag:nmmm.)

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

__tag_role_get:nnNN

```

229 \pdf_version_compare:NnF < {2.0}
230 {
231   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
232     {%#1 tag, #2 NS,
233     %#3 tlvar which hold the role tag
234     %#4 tlvar which hold the name of the target NS
235     {
236       \prop_if_exist:cTF {g__tag_role_NS_#2_prop}
237       {
238         \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
239         {
240           \tl_set:Ne #3 {\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
241           \tl_set:Ne #4 {\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
242         }
243         {
244           \msg_warning:nnn { tag } {role-unknown-tag} { #1 }
245           \tl_set:Nn #3 {#1}
246           \tl_set:Nn #4 {#2}
247         }
248       }
249       {
250         \msg_warning:nnn { tag } {role-unknown-NS} { #2 }
251         \tl_set:Nn #3 {#1}
252         \tl_set:Nn #4 {#2}
253       }
254     }
255   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
256 }

```

(End of definition for __tag_role_get:nnNN.)

1.4 Helper command to read the data from files

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

```

257 \bool_new:N\l__tag_role_update_bool
258 \bool_set_true:N \l__tag_role_update_bool
259 \pdf_version_compare:NnTF < {2.0}
260 {
261   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
262     % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
263     {
264       \tl_if_empty:nF { #2 }
265       {
266         \bool_if:NTF \l__tag_role_update_bool
267         {
268           \tl_if_empty:nTF {#5}
269           {
270             \prop_get:NnN \g__tag_role_tags_class_prop {#3}\l__tag_tmpa_tl
271             \quark_if_no_value:NT \l__tag_tmpa_tl
272             {
273               \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
274             }
275           }
276           {
277             \tl_set:Nn \l__tag_tmpa_tl {#5}
278           }
279           \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
280           \tl_if_eq:nnF {#2}{#3}
281           {
282             \__tag_role_add_tag:nn {#2}{#3}
283           }
284           \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{#3}{}}
285         }
286         {
287           \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{#3}{}}
288           \prop_gput:cnn {g__tag_role_NS_#1_class_prop} {#2}{--UNUSED--}
289         }
290       }
291     }
292 }
293 {
294   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
295     % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
296     {
297       \tl_if_empty:nF {#2}
298       {
299         \tl_if_empty:nTF {#5}
300         {
301           \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
302           \quark_if_no_value:NT \l__tag_tmpa_tl

```

```

303     {
304       \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
305     }
306   }
307   {
308     \tl_set:Nn \l__tag_tmpa_tl {#5}
309   }
310   \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
311   \bool_lazy_and:nnT
312     { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
313     {
314       \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
315     }
316   \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#3}{#4}
317 }
318 }
319 }

```

(End of definition for __tag_role_read_namespace_line:nw.)

__tag_role_read_namespace:nn This command reads a namespace file in the format tagpdf-ns-XX.def

```

320 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
321   {
322     \prop_if_exist:cF {g__tag_role_NS_#1_prop}
323       { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
324     \file_if_exist:nTF { tagpdf-ns-#2.def }
325     {
326       \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
327       \msg_info:nnn {tag}{read-namespace}{#2}
328       \ior_map_inline:Nn \g_tmpa_ior
329       {
330         \__tag_role_read_namespace_line:nw {#1} ##1,,, \q_stop
331       }
332       \ior_close:N\g_tmpa_ior
333     }
334     {
335       \msg_info:nnn{tag}{namespace-missing}{#2}
336     }
337   }
338

```

(End of definition for __tag_role_read_namespace:nn.)

__tag_role_read_namespace:n This command reads the default namespace file.

```

339 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
340   {
341     \__tag_role_read_namespace:nn {#1}{#1}
342   }

```

(End of definition for __tag_role_read_namespace:n.)

1.5 Reading the default data

The order is important as we want pdf2 and latex as default: if two namespace define the same tag, the last one defines which one is used if the namespace is not explicitly given.

```

343 \__tag_role_read_namespace:n {pdf}
344 \__tag_role_read_namespace:n {pdf2}
345 \__tag_role_read_namespace:n {mathml}

```

in pdf 1.7 the following namespaces should only store the settings for later use:

```

346 \bool_set_false:N\l__tag_role_update_bool
347 \__tag_role_read_namespace:n {latex-book}
348 \bool_set_true:N\l__tag_role_update_bool
349 \__tag_role_read_namespace:n {latex}
350 \__tag_role_read_namespace:nn {latex} {latex-lab}
351 \__tag_role_read_namespace:n {pdf}
352 \__tag_role_read_namespace:n {pdf2}

```

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

```

353 \pdf_version_compare:NnTF < {2.0}
354 {
355   \hook_gput_code:nnn {begindocument}{tagpdf}
356   {
357     \cs_if_exist:NT \chapter
358     {
359       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
360       {
361         \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
362       }
363     }
364   }
365 }
366 {
367   \hook_gput_code:nnn {begindocument}{tagpdf}
368   {
369     \cs_if_exist:NT \chapter
370     {
371       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
372       {
373         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ latex-book }
374         \prop_gput:Nne
375         \g__tag_role_rolemap_prop {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
376       }
377     }
378   }
379 }

```

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.

```

380 \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!

```
381 \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
382   {
383     \intarray_gset:Nnn \g__tag_role_parent_child_intarray
384       { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
385   }
```

(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

```
386 \int_zero:N \l__tag_tmpa_int
```

Open the file depending on the PDF version

```
387 \pdf_version_compare:NnTF < {2.0}
388   {
389     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
390   }
391   {
392     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
393   }
```

Now the main loop over the file

```
394 \ior_map_inline:Nn \g_tmpa_ior
395   {
```

ignore lines containing only comments

```
396   \tl_if_empty:nF{#1}
397     {
```

count the lines ...

```
398       \int_incr:N\l__tag_tmpa_int
```

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

```
399       \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
400       \int_compare:nNnTF {\l__tag_tmpa_int}=1
```

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

```
401     {
402       \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
403       {
404         \prop_gput:Nne\g__tag_role_index_prop
405           {##2}
406           {\int_compare:nNnT{##1}<{10}{0}##1}
407       }
408     }
```

now the data lines.

```

409     {
410         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
get the name of the child tag from the first column
411         \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
412         \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmpb_tl
remove column 2+3
413         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
414         \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.
415         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
416         {
417             \exp_args:Nne
418             \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
419         }
420     }
421 }
422 }

```

close the read handle.

```

423 \ior_close:N\g_tmpa_ior

```

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

```

424 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
425 \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
426 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
427 \pdf_version_compare:NnTF < {2.0}
428 {
429     \int_step_inline:nn{6}
430     {
431         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
432     }
433 }
434 {
435     \int_step_inline:nn{10}
436     {
437         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
438     }

```

all mathml tags are currently handled identically

```

439     \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
440     \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
441     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
442     {
443         \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
444     }
445     \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
446 }

```

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.

```

447 \tl_new:N \l__tag_parent_child_check_tl
448 \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
449   % #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
450   % #4 tl for state
451   {
452     \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
453     \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
454     \bool_lazy_and:nnTF
455     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
456     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
457     {

```

Get the rule from the intarray

```

458     \tl_set:Nc#4
459     {
460       \intarray_item:Nn
461       \g__tag_role_parent_child_intarray
462       {\l__tag_tmpa_tl\l__tag_tmpb_tl}
463     }

```

If the state is something is wrong ...

```

464     \int_compare:nNnT
465     {#4} = {\prop_item:Nn\c__tag_role_rules_prop{}}
466     {
467       %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.

```

468     }

```

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

```

469     \group_begin:
470     \int_compare:nNnT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
471     {
472       \prop_get:NVNF\c__tag_role_rules_num_prop #4 \l__tag_tmpa_tl
473       {
474         \tl_set:Nn \l__tag_tmpa_tl {unknown}
475       }
476       \tl_set:Nn \l__tag_tmpb_tl {#1}
477       \msg_note:nnee
478       { tag }
479       { role-parent-child }
480       { #1 }
481       { #2 }
482       {
483         #4~(=\l__tag_tmpa_tl')
484         \iow_newline:
485         #3

```

```

486         }
487     }
488     \group_end:
489 }
490 {
491     \tl_set:Nn#4 {0}
492     \msg_warning:nneee
493     { tag }
494     {role-parent-child}
495     { #1 }
496     { #2 }
497     { unknown! }
498 }
499 }
500 \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:nnnn

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.

```

501 \pdf_version_compare:NnTF < {2.0}
502 {
503     \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
504     {%#1 parent tag, #2 NS, #3 child tag, #4 NS, #5 tl var
505     {

```

for debugging messages we store the arguments.

```

506     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
507     \prop_put:Nnn \l__tag_role_debug_prop {child} {#3}

```

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

```

508     \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
509     {
510         \tl_set:Nn \l__tag_tmpa_tl {#1}
511     }
512     {
513         \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
514         {
515             \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
516         }
517     }

```

now the child

```

518     \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
519     {
520         \tl_set:Nn \l__tag_tmpb_tl {#3}
521     }
522     {
523         \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
524         {
525             \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
526         }
527     }

```

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

```

528     \bool_lazy_and:nnTF
529     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
530     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
531     {
532         \__tag_role_get_parent_child_rule:VVnN
533         \l__tag_tmpa_tl \l__tag_tmpb_tl
534         {Rolemapped~from:~'#1'~-->~'#3'}
535         #5
536     }
537     {
538         \tl_set:Nn #5 {0}
539         \msg_warning:nneee
540         { tag }
541         {role-parent-child}
542         { #1 }
543         { #3 }
544         { unknown! }
545     }
546 }
547 \cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
548 {
549     \__tag_check_parent_child:nnnnN {#1}{#2}{#3}
550 }
551 }

```

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.

```

552 {
553     \cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
554     {
555         \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
556         \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl
557         \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
558         \bool_lazy_and:nnTF
559         { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl }
560         { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl }
561         {
562             \__tag_check_parent_child:nVnVN
563             {#1}\l__tag_role_tag_namespace_tmpa_tl
564             {#2}\l__tag_role_tag_namespace_tmpb_tl
565             #3
566         }
567         {
568             \tl_set:Nn #3 {0}
569             \msg_warning:nneee
570             { tag }
571             {role-parent-child}
572             { #1 }
573             { #2 }
574             { unknown! }
575         }
576     }

```

and now the real command.

```
577 \cs_new_protected:Npn \__tag_check_parent_child:nmmmN #1 #2 #3 #4 #5 %tag,NS,tag,NS, t1 va
578 {
579   \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
580   \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

```
581   \tl_if_empty:nTF {#2}
582   {
583     \tl_set:Nn \l__tag_tmpa_tl {#1}
584   }
585   {
586     \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
587     {
588       \prop_get:cnNTF
589       { g__tag_role_NS_#2_prop }
590       {#1}
591       \l__tag_tmpa_tl
592       {
593         \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
594         \tl_if_empty:NT\l__tag_tmpa_tl
595         {
596           \tl_set:Nn \l__tag_tmpa_tl {#1}
597         }
598       }
599       {
600         \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
601       }
602     }
603     {
604       \msg_warning:nnn { tag } {role-unknown-NS} { #2}
605       \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
606     }
607   }
```

and the same for the child If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```
608   \tl_if_empty:nTF {#4}
609   {
610     \tl_set:Nn \l__tag_tmpb_tl {#3}
611   }
612   {
613     \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
614     {
615       \prop_get:cnNTF
616       { g__tag_role_NS_#4_prop }
617       {#3}
618       \l__tag_tmpb_tl
619       {
620         \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
621         \tl_if_empty:NT\l__tag_tmpb_tl
622         {
623           \tl_set:Nn \l__tag_tmpb_tl {#3}
624         }
625       }
626     }
627     {
628       \msg_warning:nnn { tag } {role-unknown-NS} { #4}
629       \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
630     }
631   }
```

```

624         }
625     }
626     {
627         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
628     }
629 }
630 {
631     \msg_warning:nnn { tag } {role-unknown-NS} { #4}
632     \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
633 }
634 }

```

and now get the relation

```

635     \bool_lazy_and:nnTF
636     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
637     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
638     {
639         \__tag_role_get_parent_child_rule:VVnN
640         \l__tag_tmpa_tl \l__tag_tmpb_tl
641         {Rolemapped~from~'#1/#2'~-->~'#3\str_if_empty:nF{#4}{/#4}'
642         #5
643     }
644     {
645         \tl_set:Nn #5 {0}
646         \msg_warning:nneee
647         { tag }
648         {role-parent-child}
649         { #1 }
650         { #3 }
651         { unknown! }
652     }
653 }
654 }
655 \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
656 \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
657 \end{package}

```

(End of definition for `__tag_check_parent_child:nnnnN`.)

`\tag_check_child:nnTF`

```

658 (base) \prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true:
659 (*package)
660 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}
661 {
662     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
663     \__tag_struct_get_parentrole:eNN
664     {\l__tag_tmpa_tl}
665     \l__tag_get_parent_tmpa_tl
666     \l__tag_get_parent_tmpb_tl
667     \__tag_check_parent_child:VVnnN
668     \l__tag_get_parent_tmpa_tl
669     \l__tag_get_parent_tmpb_tl
670     {#1}{#2}
671     \l__tag_parent_child_check_tl
672     \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}

```

```

673     {\prg_return_false;}
674     {\prg_return_true;}
675 }

```

(End of definition for `\tag_check_child:nTF`. This function is documented on page 154.)

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
676 \tl_new:N \l__tag_role_remap_tag_tl
677 \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?

```

678 \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.

```

679 \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:

```

(End of definition for `__tag_role_remap_id:.`)

1.8 Key-val user interface

The user interface uses the key `add-new-tag`, which takes either a keyval list as argument, or a tag/role.

```

tag_(rolemap-key)
tag-namespace_(rolemap-key)
role_(rolemap-key)
role-namespace_(rolemap-key)
role/new-tag_(setup-key)
add-new-tag_(deprecated)
680 \keys_define:nn { __tag / tag-role }
681 {
682   ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
683   ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
684   ,role .tl_set:N = \l__tag_role_role_tmpa_tl
685   ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
686 }
687
688 \keys_define:nn { __tag / setup }
689 {
690   role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
691   ,role/new-tag .code:n =
692   {
693     \keys_set_known:nnnN
694     {__tag/tag-role}
695     {

```

```

696     tag-namespace=user,
697     role-namespace=, %so that we can test for it.
698     #1
699     }{__tag/tag-role}\l_tmpa_tl
700 \tl_if_empty:NF \l_tmpa_tl
701 {
702     \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
703     \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
704     \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
705 }
706 \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
707 {
708     \prop_get:NVNTF
709     \g__tag_role_tags_NS_prop
710     \l__tag_role_role_tmpa_tl
711     \l__tag_role_role_namespace_tmpa_tl
712     {
713         \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
714         {
715             \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
716         }
717     }
718     {
719         \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
720     }
721 }
722 \pdf_version_compare:NnTF < {2.0}
723 {
724     %TODO add check for emptyness?
725     \__tag_role_add_tag:VV
726     \l__tag_role_tag_tmpa_tl
727     \l__tag_role_role_tmpa_tl
728 }
729 {
730     \__tag_role_add_tag:VVVV
731     \l__tag_role_tag_tmpa_tl
732     \l__tag_role_tag_namespace_tmpa_tl
733     \l__tag_role_role_tmpa_tl
734     \l__tag_role_role_namespace_tmpa_tl
735 }
736 }
737 ,role/map-tags .choice:
738 ,role/map-tags/false .code:n = { \socket_assign_plug:n { tag/struct/tag } {latex-
tags} }
739 ,role/map-tags/pdf .code:n = { \socket_assign_plug:n { tag/struct/tag } {pdf-
tags} }
depreciated names
740 , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
741 , add-new-tag .meta:n = {role/new-tag={#1}}
742 }
743 \end{package}

```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 154.)

Part X

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

`activate/space_ (setup-key)`
`interwordspace_ (deprecated)`

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`. The old name of the key `interwordspace` is still supported but deprecated.

`show-spaces_ (deprecated)`

This key is deprecated. Use `debug/show=spaces` instead. This key works only with `luatex` and shows with small red bars where spaces have been inserted. This is only for debugging and is not completely reliable (and change affect other literals and tagging), so it should be used with care.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2024-02-29} {0.98x}
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.

`activate/spaces_ (setup-key)`
`interwordspace_ (deprecated)`
`show-spaces_ (deprecated)`

```
6 <*package>
7 \keys_define:nn { __tag / setup }
8 {
9   activate/spaces .choice:,
10  activate/spaces/true .code:n =
11    { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
12  activate/spaces/false .code:n=
13    { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
14  activate/spaces .default:n = true,
15  debug/show/spaces .code:n = {\bool_set_true:N \l__tag_showspaces_bool},
16  debug/show/spacesOff .code:n = {\bool_set_false:N \l__tag_showspaces_bool},
```

deprecated versions:

```
17   interwordspace .choices:nn = {true,on}{\keys_set:nn{__tag/setup}{activate/spaces={true}}},
18   interwordspace .choices:nn = {false,off}{\keys_set:nn{__tag/setup}{activate/spaces={false}}},
19   interwordspace .default:n = {true},
20   show-spaces .choice:,
```

```

21   show-spaces/true .meta:n = {debug/show=spaces},
22   show-spaces/false .meta:n = {debug/show=spacesOff},
23   show-spaces .default:n = true
24 }
25 \sys_if_engine_pdftex:T
26 {
27   \sys_if_output_pdf:TF
28   {
29     \pdfglyphtounicode{space}{0020}
30     \keys_define:nn { __tag / setup }
31     {
32       activate/spaces/true .code:n = { \pdfinterwordspaceon },
33       activate/spaces/false .code:n = { \pdfinterwordspaceoff },
34       activate/spaces .default:n = true,
35     }
36   }
37   {
38     \keys_define:nn { __tag / setup }
39     {
40       activate/spaces .choices:nn = { true, false }
41       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
42       activate/spaces .default:n = true,
43     }
44   }
45 }
46
47
48 \sys_if_engine_luatex:T
49 {
50   \keys_define:nn { __tag / setup }
51   {
52     activate/spaces .choice:,
53     activate/spaces/true .code:n =
54     {
55       \bool_gset_true:N \g__tag_active_space_bool
56       \lua_now:e{!tx.__tag.func.markspaceon()}
57     },
58     activate/spaces/false .code:n =
59     {
60       \bool_gset_false:N \g__tag_active_space_bool
61       \lua_now:e{!tx.__tag.func.markspaceoff()}
62     },
63     activate/spaces .default:n = true,
64     debug/show/spaces .code:n =
65     { \lua_now:e{!tx.__tag.trace.showspace=true} },
66     debug/show/spacesOff .code:n =
67     { \lua_now:e{!tx.__tag.trace.showspace=nil} },
68   }
69 }

```

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

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

```
70 \sys_if_engine_luatex:T
71 {
72   \cs_new_protected:Nn \__tag_fakespace:
73   {
74     \group_begin:
75     \lua_now:e{!tx.__tag.func.fakespace()}
76     \skip_horizontal:n{\c_zero_skip}
77     \group_end:
78   }
79 }
80 </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

`\#` 1034, 1038
`\\` . 10, 23, 27, 28, 44, 45, 46, 53, 56, 58,
64, 66, 77, 256, 257, 258, 396, 459, 467
`_` 421, 432

A

`activate_`(setup-key) 35, 255
`activate-all_`(deprecated) 6
`activate-mc_`(deprecated) 6
`activate-struct_`(deprecated) 6
`activate-tree_`(deprecated) 6
`activate/all_`(setup-key) 6, 237
`activate/mc_`(setup-key) 6, 237
`activate/socket_`(setup-key) 255
`activate/space_`(setup-key) 175
`activate/spaces_`(setup-key) 6, 6
`activate/struct_`(setup-key) 6, 237
`activate/struct-dest_`(setup-key) 6, 237
`activate/tagunmarked_`(setup-key) 6, 268
`activate/tree_`(setup-key) 6, 237
`actualtext_`(mc-key) 69, 255, 453
`actualtext_`(struct-key) 100, 504
`add-new-tag_`(deprecated) 680
`add-new-tag_`(setup-key) 154
`\AddToHook` 13, 16, 57, 87, 273, 356, 378,
509, 511, 512, 516, 520, 527, 556, 601
`AF_`(struct-key) 100, 615
`AFinline_`(struct-key) 100, 615
`AFinline-o_`(struct-key) 100, 615
`AFref_`(struct-key) 100, 615
`alt_`(mc-key) 69, 255, 453
`alt_`(struct-key) 99, 504
`artifact_`(mc-key) 69, 255, 453
artifact-bool internal commands:
 `__artifact-bool` 121
artifact-type internal commands:
 `__artifact-type` 121
`attr-unknown` 20, 70
`attribute_e_`(struct-key) 101, 1186
`attribute-class_`(struct-key) . 101, 1152

B

bool commands:
 `\bool_gset_eq:NN` ... 621, 636, 648, 666
 `\bool_gset_false:N`
 50, 51, 60, 238, 441, 622, 649

`\bool_gset_true:N`
 47, 49, 55, 110, 177, 369
`\bool_if:NTF` . 9, 13, 18, 27, 36, 40,
66, 69, 74, 79, 114, 192, 200, 210,
223, 234, 238, 255, 266, 278, 289,
298, 304, 339, 354, 373, 390, 418,
423, 429, 441, 443, 460, 468, 493,
500, 560, 616, 631, 643, 661, 841, 901
`\bool_if:nTF` 6, 346
`\bool_if_exist_p:N` 44
`\bool_lazy_all:nTF` 102
`\bool_lazy_and:nnTF` . 43, 136, 146,
275, 311, 366, 454, 510, 528, 558, 635
`\bool_lazy_and_p:nn` 8
`\bool_new:N` 16, 20, 21, 41,
42, 63, 73, 105, 106, 107, 108, 109,
111, 113, 115, 116, 257, 293, 294, 612
`\bool_set_false:N`
 16, 178, 188, 189, 190, 211,
212, 213, 239, 346, 390, 593, 615, 642
`\bool_set_true:N`
 15, 112, 114, 198, 199,
200, 221, 222, 223, 258, 348, 389, 592
`\box` 375
box commands:
 `\box_dp:N` 176, 180
 `\box_ht:N` 166
 `\box_new:N` 100, 101
 `\box_set_dp:Nn` 174, 176
 `\box_set_eq:NN` 189
 `\box_set_ht:Nn` 173, 175
 `\box_use_drop:N` 178, 182
`\boxmaxdepth` 80, 177

C

`\c` 269, 270
`c@g` internal commands:
 `\c@g__tag_MCID_abs_int` . 11, 15, 24,
33, 46, 53, 64, 70, 80, 134, 159, 180,
239, 242, 269, 274, 303, 344, 351, 416
 `\c@g__tag_parenttree_obj_int` 151, 455
 `\c@g__tag_struct_abs_int` . 6, 18,
38, 81, 89, 112, 113, 145, 147, 150,
152, 229, 354, 480, 515, 537, 549,
563, 579, 587, 600, 610, 629, 632,
637, 671, 673, 678, 690, 692, 697,
750, 761, 762, 763, 764, 765, 766,
769, 771, 777, 780, 797, 804, 822,

831, 839, 867, 875, 880, 895, 896, 898, 909, 1007, 1070, 1179, 1182, 1230	
cctab commands:	
\c_document_cctab	73
\chapter	165, 357, 369
clist commands:	
\clist_const:Nn	102, 103
\clist_if_empty:NTF	1191
\clist_map_inline:nn	126, 594
\clist_new:N	98
\clist_set:Nn	1156, 1190
color commands:	
\color_select:n	421, 432
cs commands:	
\cs_generate_variant:Nn	40, 77, 94, 103, 105, 118, 119, 120, 121, 122, 123, 124, 125, 126, 126, 138, 139, 140, 155, 166, 170, 175, 175, 176, 177, 177, 178, 179, 180, 192, 224, 228, 240, 255, 264, 264, 275, 321, 332, 386, 500, 616, 644, 655, 656, 664, 1086, 1095, 1110, 1120, 1140
\cs_gset_eq:NN	271, 872, 873, 1004, 1005, 1067, 1068
\cs_if_exist:NTF	357, 369, 562, 603
\cs_if_exist_p:N	9
\cs_if_exist_use:NTF	374, 1089
\cs_if_free:NTF	47
\cs_new:Nn 80, 81, 107, 129, 134, 349, 385	
\cs_new:Npn	9, 15, 26, 68, 94, 98, 138, 141, 158, 193, 231, 254, 352, 458, 466, 472, 478, 1082, 1121
\cs_new_eq:NN	127, 128, 129, 130
\cs_new_protected:Nn 72, 127, 169, 352	
\cs_new_protected:Npn	13, 19, 20, 22, 30, 30, 35, 40, 41, 59, 59, 60, 61, 62, 65, 65, 70, 74, 78, 78, 79, 80, 80, 84, 85, 87, 89, 95, 104, 107, 117, 124, 131, 140, 143, 149, 150, 154, 159, 160, 162, 163, 167, 168, 169, 176, 178, 182, 185, 192, 205, 211, 223, 225, 230, 230, 231, 232, 233, 234, 235, 236, 238, 241, 245, 253, 257, 261, 265, 267, 268, 273, 276, 276, 292, 294, 310, 310, 311, 314, 316, 318, 320, 322, 322, 326, 333, 337, 337, 339, 341, 345, 351, 356, 366, 373, 380, 381, 387, 387, 407, 415, 415, 420, 422, 426, 430, 437, 448, 503, 547, 553, 577, 596, 597, 598, 599, 613, 617, 627, 640, 645, 656, 678, 750, 751, 752, 961, 1018, 1087, 1098, 1111, 1134
\cs_set:Nn	681, 682
\cs_set:Npn	44, 49
\cs_set_eq:NN	14, 20, 76, 77, 78, 89, 165, 166, 167, 168, 169, 170, 171, 172, 173, 205, 206, 230, 231, 232, 233, 347, 348, 349, 350, 674, 675, 676, 677, 679, 683, 684, 688, 689, 690, 691, 869, 870, 1001, 1002, 1064, 1065
\cs_set_protected:Nn	171, 233, 260, 428, 434, 919, 920
\cs_set_protected:Npn	9, 15, 16, 22, 29, 36, 37, 55, 62, 62, 67, 70, 72, 77, 82, 88, 101, 141, 184, 193, 206, 207, 210, 216, 226, 237, 244, 245, 328, 332, 336, 340, 355, 358, 361, 754, 755, 955, 963, 1020
\cs_to_str:N	12, 18, 25, 32, 39, 58, 59, 65, 66
D	
debug/log _␣ (setup-key)	6, 255
debug/show _␣ (setup-key)	254
debug/structures _␣ (show-key)	36, 224
debug/uncompress _␣ (setup-key)	255
\DeclareOption	49, 50, 51
dim commands:	
\c_max_dim	165, 190
\c_zero_dim	173, 174, 175
\documentclass	22
\DocumentMetadata	21
E	
E _␣ (struct-key)	100, 504
\endinput	28
\ERRORusetaggingsocket	89
exclude-header-footer _␣ (deprecated) 669	
\ExecuteOptions	52
exp commands:	
\exp_args:Ne	118, 406, 460, 767
\exp_args:NNe	74, 82, 85, 191, 211
\exp_args:Nne 79, 340, 344, 417, 446, 492	
\exp_args:NNne	74
\exp_args:NNno	702
\exp_args:NV 196, 202, 347, 376, 387, 392	
\exp_last_unbraced:NV	184, 185, 240, 241, 445, 449, 944
\exp_not:n	308
F	
file commands:	
\file_if_exist:nTF	324
\file_input:n	285
flag commands:	
\flag_clear:n	236

<code>\flag_height:n</code>	163, 248	<code>\int_incr:N</code>	91, 187, 210, 398
<code>\flag_new:n</code>	161	<code>\int_new:N</code>	76, 99, 104, 183, 263, 296, 297, 298, 299, 615
<code>\flag_raise:n</code>	249	<code>\int_rand:n</code> ..	61, 62, 64, 66, 68, 70, 71
<code>\fontencoding</code>	6	<code>\int_set:Nn</code> ...	256, 259, 262, 263, 264
<code>\fontfamily</code>	6	<code>\int_step_inline:nn</code> ...	89, 429, 435
<code>\fontseries</code>	6	<code>\int_step_inline:nnn</code>	25, 229
<code>\fontshape</code>	6	<code>\int_step_inline:nnnn</code>	145, 170, 173, 190, 300, 306
<code>\fontsize</code>	6	<code>\int_to_arabic:n</code>	143, 145
<code>\footins</code>	565	<code>\int_to_Hex:n</code> ..	61, 62, 64, 66, 68, 70, 71
G			
group commands:			
<code>\group_begin:</code>	66, 74, 175, 367, 469, 622, 712, 720, 760	<code>\int_use:N</code>	11, 15, 18, 24, 33, 38, 46, 53, 64, 70, 73, 79, 80, 81, 83, 121, 123, 147, 150, 152, 157, 159, 186, 203, 209, 226, 231, 242, 251, 266, 274, 303, 354, 416, 421, 432, 480, 537, 538, 539, 547, 548, 549, 563, 579, 587, 600, 610, 626, 629, 632, 671, 673, 690, 692, 771, 777, 780, 804, 822, 831, 875, 880, 1007, 1070, 1121, 1179, 1230
<code>\group_end:</code>	73, 77, 230, 419, 488, 640, 716, 724, 915	<code>\int_zero:N</code>	88, 103, 386
H			
<code>\hangindent</code>	373	intarray commands:	
<code>\hbox</code>	364	<code>\intarray_gset:Nnn</code>	278, 383
hbox commands:			
<code>\hbox_set:Nn</code>	167, 168	<code>\intarray_item:Nn</code>	280, 283, 460
hook commands:			
<code>\hook_gput_code:nnn</code>	7, 11, 33, 57, 64, 78, 152, 236, 258, 259, 352, 355, 356, 367, 697, 710, 720, 733	<code>\intarray_new:Nn</code>	270, 380
<code>\hook_new:n</code>	336	interwordspace _␣ (deprecated) 175, 6	
<code>\hook_use:n</code>	341	ior commands:	
I			
<code>\ignorespaces</code>	35	<code>\ior_close:N</code>	332, 423
int commands:			
<code>\int_abs:n</code>	143	<code>\ior_map_inline:Nn</code>	328, 394
<code>\int_case:nnTF</code>	82, 290	<code>\ior_open:Nn</code>	326, 389, 392
<code>\int_compare:nNnTF</code> ..	22, 56, 68, 96, 112, 118, 123, 132, 169, 171, 172, 201, 211, 220, 247, 250, 275, 281, 362, 368, 375, 382, 389, 400, 401, 406, 409, 417, 424, 432, 439, 464, 470, 533, 542, 672, 788, 854, 999, 1062	<code>\g_tmpa_ior</code>	326, 328, 332, 389, 392, 394, 423
<code>\int_compare:nTF</code>	176, 315, 1172, 1174, 1176, 1200, 1226	iow commands:	
<code>\int_compare_p:nNn</code>	515	<code>\iow_newline:</code>	201, 293, 484
<code>\int_decr:N</code>	195, 218	<code>\iow_now:Nn</code>	74
<code>\int_eval:n</code>	134, 187, 291, 308, 361, 460, 468, 512, 517, 520, 637, 678, 697, 762, 763, 764, 765, 766, 769, 867, 895, 896, 898, 909, 1182	<code>\iow_term:n</code> ..	181, 184, 190, 194, 194, 260
<code>\int_gincr:N</code>	180, 239, 269, 312, 316, 320, 324, 330, 334, 338, 342, 344, 351, 455, 623, 750, 761	K	
<code>\int_gset:Nn</code>	80, 154, 287	kernel internal commands:	
<code>\int_gzero:N</code>	7, 295	<code>_kernel_pdffdict_name:n</code>	43
<code>\int_if_zero:nTF</code>	195, 196, 218, 219, 456, 464	keys commands:	
		<code>\keys_define:nn</code>	7, 30, 32, 38, 50, 99, 111, 121, 173, 216, 225, 238, 255, 261, 386, 395, 402, 408, 454, 504, 665, 669, 680, 688, 727, 1141, 1152, 1186
		<code>\keys_set:nn</code>	10, 17, 18, 18, 96, 187, 266, 341, 345, 372, 493, 775
		<code>\keys_set_known:nnnN</code>	693
L			
<code>label_␣(mc-key)</code>	69, 255, 453	<code>label_␣(struct-key)</code>	99, 504
<code>lang_␣(struct-key)</code>	100, 504		

legacy commands:	
\legacy_if:nTF	72, 462, 465, 466
\llap	421
log_(deprecated)	255
ltx. internal commands:	
ltx.__tag.func.alloctag	271
ltx.__tag.func.fakespace	445
ltx.__tag.func.fill_parent_tree_- line	812
ltx.__tag.func.get_num_from . . .	280
ltx.__tag.func.get_tag_from . . .	299
ltx.__tag.func.mark_page_- elements	643
ltx.__tag.func.mark_shipout . . .	795
ltx.__tag.func.markspaceoff . . .	511
ltx.__tag.func.markspaceon . . .	511
ltx.__tag.func.mc_insert_kids . .	580
ltx.__tag.func.mc_num_of_kids . .	329
ltx.__tag.func.output_num_from .	280
ltx.__tag.func.output_parenttree	812
ltx.__tag.func.output_tag_from .	299
ltx.__tag.func.pdf_object_ref . .	425
ltx.__tag.func.space_chars_- shipout	543
ltx.__tag.func.store_mc_data . .	314
ltx.__tag.func.store_mc_in_page	624
ltx.__tag.func.store_mc_kid . . .	323
ltx.__tag.func.store_mc_label . .	319
ltx.__tag.func.store_struct_- mcabs	612
ltx.__tag.func.update_mc_- attributes	632
ltx.__tag.tables.role_tag_- attribute	269
ltx.__tag.trace.log	183
ltx.__tag.trace.show_all_mc_data	240
ltx.__tag.trace.show_mc_data . .	225
ltx.__tag.trace.show_prop	200
ltx.__tag.trace.show_seq	191
ltx.__tag.trace.show_struct_data	246
lua commands:	
\lua_now:n	8, 12, 15, 18, 25, 26, 32, 35, 39, 42, 46, 50, 51, 56, 58, 59, 59, 61, 65, 65, 66, 67, 71, 75, 84, 85, 87, 94, 105, 109, 109, 118, 122, 130, 131, 136, 142, 156, 183, 227, 247, 261, 269, 285, 306, 320, 330
M	
mathml	100
\maxdimen	188
mc-current	19, 16
mc-current_(show-key)	36, 111
mc-data_(show-key)	36, 99
mc-label-unknown	19, 9
mc-marks_(show-key)	36, 173
mc-nested	19, 6
mc-not-open	19, 13
mc-popped	19, 14
mc-pushed	19, 14
mc-tag-missing	19, 8
mc-used-twice	19, 12
\MessageBreak	15, 19, 20, 21
msg commands:	
\msg_error:nn	159, 180, 430, 794
\msg_error:nnn	196, 207, 215, 226, 261, 417, 1166, 1206
\msg_error:nnnnn	535, 544
\msg_info:nnn 134, 173, 174, 249, 253, 327, 335
\msg_info:nnnn	203, 222
\msg_line_context: 77, 363, 364, 396, 400, 404, 460, 468
\g_msg_module_name_prop	30, 34
\g_msg_module_type_prop	33
\msg_new:nnn	7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 51, 60, 71, 72, 73, 74, 75, 76, 78, 80, 81, 82, 83, 84, 85, 87, 254, 363, 364, 394, 398, 402, 454, 462
\msg_new:nnnn	90
\msg_note:nn	169
\msg_note:nnn 186, 203, 384, 391, 426, 434
\msg_note:nnnn 209, 226, 370, 377, 411, 419
\msg_note:nnnnn	477
\msg_redirect_name:nnn	531
\msg_show_item_unbraced:n	246
\msg_show_item_unbraced:nn	237
\msg_term:nnnnnn	231, 240
\msg_warning:nn	24, 212, 261
\msg_warning:nnn	11, 13, 41, 44, 53, 166, 189, 234, 242, 244, 250, 265, 288, 323, 604, 631, 1013, 1032, 1076
\msg_warning:nnnn	397, 447, 519
\msg_warning:nnnnn 217, 407, 492, 539, 569, 646, 860
N	
namespace_(rolemap-key)	154
new-tag	20, 80
newattribute_(deprecated)	101, 1134
\newcommand	589, 590
\newcounter	6, 8, 151
\NewDocumentCommand	6, 23, 29, 34, 40, 46, 51, 56, 94, 286, 594
\newmarks	13

no-struct-dest _□ (deprecated)	6	\pdf_version_compare:NnTF	20, 81, 125, 148, 156,
\noindent	373		229, 259, 312, 353, 387, 427, 501, 722
\nointerlineskip	181	pdfannot commands:	
P			
\PackageError	13	\pdfannot_dict_put:nnn	119, 704, 727, 745, 750
\PackageWarning	28, 553	\pdfannot_link_ref_last: . . .	714, 737
page/exclude-header-footer _□ (setup-		pdfdict commands:	
key)	38, 669	\pdfdict_gput:nnn	38, 45, 53, 189, 266, 322
page/tabsorder _□ (setup-key)	6, 271	\pdfdict_if_empty:nTF	316
para-flattened _□ (deprecated)	386	\pdfdict_new:n	18, 35, 37
para-hook-count-wrong	20, 90	\pdfdict_put:nnn	713, 714, 721, 722
para/flattened _□ (tool-key)	386	\pdfdict_use:n	273, 320, 327
para/maintag _□ (setup-key)	386	\pdffakespace	37, 284
para/maintag _□ (tool-key)	386	pdffile commands:	
para/tag _□ (setup-key)	386	\pdffile_embed_stream:nnN . .	616, 624
para/tag _□ (tool-key)	386	\pdffile_embed_stream:nnn	120
para/tagging _□ (setup-key)	37, 386	\pdfglyphtoupper	29
para/tagging _□ (tool-key)	386	\pdfinterwordspaceoff	33
\PARALABEL	486	\pdfinterwordspaceon	32
paratag _□ (deprecated)	386	pdfmanagement commands:	
paratagging _□ (deprecated)	37, 386	\pdfmanagement_add:nnn	50, 68, 69, 273, 275, 277, 358
paratagging-show _□ (deprecated) . .	37, 386	\pdfmanagement_if_active_p: . .	9, 10
parent _□ (struct-key)	99, 504	\pdfmanagement_remove:nn	279
pdf commands:		pdfmanagement internal commands:	
\pdf_activate_structure_destination:	281	\l_pdfmanagement_delayed_	
\pdf_bdc:nn	232	shipout_bool	44, 45
\pdf_bdc_shipout:nn	233	prg commands:	
\pdf_bmc:n	230	\prg_do_nothing:	78, 85, 271,
\l_pdf_current_structure_		347, 348, 349, 350, 688, 689, 690, 691	
destination_tl	279	\prg_generate_conditional_	
\pdf_emc:	231	variant:Nnn	117
\pdf_name_from_unicode_e:n	98, 108, 113,	\prg_new_conditional:Nnn	66, 221
	156, 165, 192, 268, 1137, 1160, 1196	\prg_new_conditional:Npnn	96, 119, 134, 144, 338, 344, 355
\pdf_object_if_exist:n	117	\prg_new_eq_conditional:NNn . .	80, 228
\pdf_object_if_exist:nTF	150, 199, 389, 669, 731	\prg_new_protected_conditional:Npnn	658
\pdf_object_new:n	100,		
	29, 34, 36, 150, 252, 298, 309, 768	\prg_replicate:nn	142
\pdf_object_ref:n	100,	\prg_return_false: 76, 97, 114, 125,	
	55, 56, 72, 94, 118, 129, 130, 133,	128, 141, 151, 225, 341, 353, 359, 673	
	152, 194, 201, 231, 306, 323, 393,	\prg_return_true: . . . 77, 111, 124,	
	452, 671, 733, 883, 980, 1043, 1084	138, 148, 224, 342, 352, 358, 658, 674	
\pdf_object_ref_last:		\prg_set_conditional:Npnn	100
	100, 102, 116, 122, 254, 1215	\prg_set_protected_conditional:Npnn	660
\pdf_object_unnamed_write:nn . . .	98, 109, 118, 246, 1210	\ProcessOptions	53
\pdf_object_write:nnn	136, 247, 271, 299, 318, 325, 330, 407	prop commands:	
\pdf_pageobject_ref:n	199, 443	\prop_clear:N	172
\pdf_string_from_unicode:nnN . . .	42	\prop_count:N	193
\pdf_uncompress:	265, 267		

<code>\prop_get:NnN</code>	137, 145, 179, 200, 213, 216, 270, 301, 403, 412, 424, 426, 439, 440, 452, 453, 555, 556, 856, 1101
<code>\prop_get:NnNTF</code>	42, 92, 160, 166, 179, 184, 195, 199, 218, 238, 279, 391, 472, 508, 513, 518, 523, 588, 615, 708, 816, 945, 1027
<code>\prop_gput:Nnn</code>	26, 30, 31, 33, 34, 56, 88, 89, 90, 91, 92, 94, 97, 97, 98, 99, 99, 100, 110, 111, 112, 113, 119, 120, 121, 121, 122, 148, 151, 163, 168, 203, 207, 219, 222, 259, 284, 287, 288, 316, 328, 373, 374, 393, 394, 404, 425, 431, 437, 443, 445, 897, 908, 982, 1045, 1136, 1215
<code>\prop_gremove:Nn</code>	136
<code>\prop_gset_eq:NN</code>	135, 894
<code>\prop_if_exist:NTF</code>	176, 236, 322, 586, 613, 967, 1024
<code>\prop_if_exist_p:N</code>	512
<code>\prop_if_in:NnTF</code>	70, 130, 156, 164, 263, 369, 713, 1164, 1204, 1208
<code>\prop_item:Nn</code>	41, 74, 123, 171, 182, 221, 284, 376, 379, 384, 463, 465, 487, 495, 1213, 1220
<code>\prop_map_function:NN</code>	235
<code>\prop_map_inline:Nn</code>	257, 262, 314, 359, 371, 441
<code>\prop_map_tokens:Nn</code>	332
<code>\prop_new:N</code>	7, 8, 9, 10, 11, 11, 19, 24, 25, 32, 33, 95, 133, 165, 763, 1130, 1133
<code>\prop_new_linked:N</code>	17, 65, 67, 166
<code>\prop_put:Nnn</code>	122, 179, 506, 507, 579, 580
<code>\prop_show:N</code>	64, 91, 173, 891, 912, 1182, 1209
property commands:	
<code>\property_gset:nynn</code>	128
<code>\property_new:nynn</code>	127
<code>\property_record:nn</code>	134
<code>\property_ref:nn</code>	99, 130
<code>\property_ref:mnn</code>	129
<code>\providecommand</code>	62, 63, 64, 291, 558, 559
<code>\ProvidesExplFile</code>	3
<code>\ProvidesExplPackage</code>	3, 3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 26, 37, 1126
Q	
<code>\quad</code>	203, 204
quark commands:	
<code>\q_no_value</code>	515, 525, 600, 605, 627, 632
<code>\quark_if_no_value:NTF</code>	138, 146, 180, 201, 217, 271, 302, 1108
<code>\quark_if_no_value_p:N</code>	455, 456, 529, 530, 559, 560, 636, 637
<code>\q_stop</code>	261, 294, 330
R	
<code>raw_(mc-key)</code>	69, 255, 453
<code>ref_(struct-key)</code>	100, 504
regex commands:	
<code>\regex_replace_once:nnN</code>	268
<code>\RemoveFromHook</code>	514, 515
<code>\renewcommand</code>	592, 593
<code>\RenewDocumentCommand</code>	8
<code>\RequirePackage</code>	20, 54, 291, 294, 300, 303, 554
<code>\rlap</code>	432
<code>role_(rolemap-key)</code>	154, 680
<code>role-missing</code>	20, 72
<code>role-namespace_(rolemap-key)</code>	154, 680
<code>role-parent-child</code>	20, 76
<code>role-remapping</code>	20, 78
<code>role-tag</code>	20, 80
<code>role-unknown</code>	20, 72
<code>role-unknown-NS</code>	20, 72
<code>role-unknown-tag</code>	20, 72
<code>role/new-attribute_(setup-key)</code>	101, 1134
<code>role/new-tag_(setup-key)</code>	680
<code>root-AF_(setup-key)</code>	101, 727
S	
<code>\selectfont</code>	6
seq commands:	
<code>\seq_clear:N</code>	280, 305
<code>\seq_const_from_clist:Nn</code>	21, 34
<code>\seq_count:N</code>	22, 25, 56, 292, 401, 1172, 1174, 1176, 1200, 1226
<code>\seq_get:NN</code>	662
<code>\seq_get:NNTF</code>	426, 441, 790, 934, 941
<code>\seq_gpop:NN</code>	927
<code>\seq_gpop:NNTF</code>	105, 928
<code>\seq_gpop_left:NN</code>	267
<code>\seq_gpush:Nn</code>	13, 15, 88, 95, 797, 837
<code>\seq_gput_left:Nn</code>	272, 1168
<code>\seq_gput_right:Nn</code>	38, 144, 150, 169, 213, 233, 256, 325
<code>\seq_gremove_duplicates:N</code>	279
<code>\seq_gset_eq:NN</code>	155, 217, 287
<code>\seq_if_empty:NTF</code>	196, 395
<code>\seq_item:Nn</code>	57, 112, 114, 121, 125, 132, 136, 170, 309, 316, 329, 348, 350, 357, 488, 489, 496, 497, 703, 704
<code>\seq_log:N</code>	171, 195, 219, 254, 412, 427
<code>\seq_map_function:NN</code>	244
<code>\seq_map_indexed_inline:Nn</code>	402, 415
<code>\seq_map_inline:Nn</code>	281, 359, 1162, 1202

<code>\tag_start:n</code>	<code>\g__tag_attr_objref_prop</code>
... 6, 72, 182, 216, 235, 374, 629, 658 1129, 1208, 1215, 1220
<code>\tag_stop:</code> ... 6, 47, 182, 184, 205, 230	<code>\l__tag_attr_value_tl</code> 1129,
<code>\tag_stop:n</code>	1198, 1217, 1222, 1224, 1228, 1232
... 6, 67, 182, 207, 234, 372, 625, 653	<code>__tag_backend_create_bdc_node</code> .. 395
<code>\tag_struct_begin:n</code>	<code>__tag_backend_create_bmc_node</code> .. 366
..... 98, 48, 446, 453, 471,	<code>__tag_backend_create_emc_node</code> .. 337
481, 651, 702, 725, 750, 750, 754, 755	<code>__tag_check_add_tag_role:nn</code> ...
<code>\tag_struct_end:</code> 98, 26, 53, 499, 503, 129, 192, 192
660, 716, 739, 750, 751, 919, 920, 958	<code>__tag_check_add_tag_role:nnn</code> ...
<code>\tag_struct_end:n</code> 98, 752, 955 171, 211
<code>\tag_struct_gput:nnn</code>	<code>__tag_check_if_active_mc:</code> 134
..... 99, 1087, 1087, 1095	<code>__tag_check_if_active_mc:TF</code> ...
<code>\tag_struct_insert_annot:nn</code> 84, 103,
.. 98, 128, 714, 737, 1111, 1111, 1120	133, 173, 187, 235, 357, 363, 430, 436
<code>\tag_struct_object_ref:n</code>	<code>__tag_check_if_active_struct:</code> . 144
..... 98, 1081, 1082, 1086	<code>__tag_check_if_active_struct:TF</code>
<code>\tag_struct_parent_int:</code> 98, 39, 133,
128, 707, 714, 730, 737, 1111, 1121	757, 758, 924, 925, 957, 965, 1022, 1114
<code>\tag_struct_use:n</code>	<code>__tag_check_if_mc_in_galley:</code> .. 338
..... 98, 99, 58, 961, 961, 963	<code>__tag_check_if_mc_in_galley:TF</code> .
<code>\tag_struct_use_num:n</code> 179, 200
..... 98, 1018, 1018, 1020	<code>__tag_check_if_mc_tmb_missing:</code> 344
<code>\tag_tool:n</code> 35, 13, 13, 14, 16, 20	<code>__tag_check_if_mc_tmb_missing:TF</code>
tag internal commands: 108, 188, 205, 344
<code>__tag_activate_mark_space</code> 511	<code>__tag_check_if_mc_tmb_missing_-</code>
<code>\g__tag_active_mc_bool</code>	p: 344
..... 40, 105, 105, 136, 240, 247	<code>__tag_check_if_mc_tme_missing:</code> 355
<code>\l__tag_active_mc_bool</code>	<code>__tag_check_if_mc_tme_missing:TF</code>
.... 108, 111, 136, 189, 199, 212, 222 151, 192, 209, 355
<code>\l__tag_active_socket_bool</code> .. 69,	<code>__tag_check_if_mc_tme_missing_-</code>
74, 79, 111, 190, 200, 213, 223, 263	p: 355
<code>\g__tag_active_space_bool</code>	<code>__tag_check_info_closing_-</code>
..... 13, 55, 60, 105	struct:n 169, 169, 177, 930
<code>\g__tag_active_struct_bool</code>	<code>__tag_check_init_mc_used:</code>
.... 104, 105, 146, 242, 249, 277, 423 268, 268, 271, 277
<code>\l__tag_active_struct_bool</code>	<code>__tag_check_mc_if_nested:</code>
.... 107, 111, 146, 188, 198, 211, 221 176, 230, 230, 368
<code>\g__tag_active_struct_dest_bool</code> .	<code>__tag_check_mc_if_open:</code>
..... 105, 246, 253, 276 230, 237, 238, 440
<code>\g__tag_active_tree_bool</code>	<code>__tag_check_mc_in_galley:TF</code> .. 338
... 9, 66, 105, 106, 241, 248, 339, 354	<code>__tag_check_mc_in_galley_p:</code> .. 338
<code>__tag_add_missing_mcs:Nn</code>	<code>__tag_check_mc_pushed_popped:nn</code>
..... 81, 163, 163, 215 89, 96, 109, 112, 117, 245, 245
<code>__tag_add_missing_mcs_to_-</code>	<code>__tag_check_mc_tag:N</code>
stream:Nn 65, 189, 257, 257, 380
65, 185, 185, 565, 569, 574, 581, 583	<code>__tag_check_mc_used:n</code>
<code>\g__tag_attr_class_used_seq</code> 143, 273, 273, 324
..... 279, 280, 1129, 1168	<code>\g__tag_check_mc_used_intarray</code> ..
<code>\g__tag_attr_entries_prop</code> 268, 278, 280, 283
285, 1129, 1136, 1164, 1204, 1209, 1213	<code>__tag_check_no_open_struct:</code> ...
<code>__tag_attr_new_entry:nn</code> 178, 178, 932, 939
... 639, 1134, 1134, 1140, 1145, 1149	<code>__tag_check_para_begin_show:nn</code> .
 415, 454, 486

__tag_check_para_end_show:nn ...	426, 498	__tag_get_data_struct_id: .	466, 466
__tag_check_parent_child:nnN ...	547, 553, 655	__tag_get_data_struct_num:	471, 472
__tag_check_parent_child:nnnnN .	501	__tag_get_data_struct_tag:	458, 458
__tag_check_parent_child:nnnnN .	206, 396, 503, 549, 562, 577, 656, 667, 848, 993, 1056	__tag_get_mathsubtype	261
__tag_check_show_MCID_by_page: .	292, 292	__tag_get_mc_abs_cnt:	14, 15, 19, 20, 100, 105, 135, 146, 185, 227, 234, 242, 261, 263, 271, 289, 310, 324, 334
__tag_check_struct_used:n	182, 182, 970	__tag_get_mc_cnt_type_tag	255
__tag_check_structure_has_tag:n	154, 154, 780	__tag_get_num_from	280
__tag_check_structure_tag:N	162, 162, 490, 501	\l__tag_get_parent_tmpa_tl	92, 204, 207, 220, 394, 397, 410, 665, 668, 846, 849, 863, 905
__tag_check_timeout_v:n	89, 89, 106, 107, 110, 145, 153, 160, 198, 207, 260, 464, 480, 496, 568, 573, 578	\l__tag_get_parent_tmpa_tl_uuuu\l__tag_get_parent_tmpb_tl_uuuu\l__tag_tmpa_str	89
__tag_debug_mc_begin_ignore:n	373, 423	\l__tag_get_parent_tmpb_tl	93, 205, 208, 220, 395, 398, 410, 666, 669, 847, 850, 863
__tag_debug_mc_begin_insert:n	365, 366	__tag_get_tag_from	299
__tag_debug_mc_end_ignore:	387, 448	\l__tag_get_tmpe_tl	89, 166, 171, 182, 184, 185, 238, 240, 241, 819, 825, 1104, 1108
__tag_debug_mc_end_insert:	380, 438	__tag_gincr_para_begin_int:	310, 314, 332, 348, 361, 452, 479
__tag_debug_struct_begin_ignore:n	415, 917	__tag_gincr_para_end_int:	310, 322, 340, 350, 495
__tag_debug_struct_begin_insert:n	407, 914	__tag_gincr_para_main_begin_int:	310, 310, 328, 347, 445, 470
__tag_debug_struct_end_check:n	437, 957	__tag_gincr_para_main_end_int:	310, 318, 336, 349, 502
__tag_debug_struct_end_ignore:	430, 952	__tag_hook_kernel_after_foot:	599, 608, 677, 684, 691
__tag_debug_struct_end_insert:	422, 950	__tag_hook_kernel_after_head:	597, 606, 676, 683, 690
\g__tag_delayed_shipout_bool	42, 47, 51, 234	__tag_hook_kernel_before_foot:	598, 607, 675, 682, 689
__tag_exclude_headfoot_begin:	613, 674, 675	__tag_hook_kernel_before_head:	596, 605, 674, 681, 688
__tag_exclude_headfoot_end:	627, 676, 677	\g__tag_in_mc_bool	16, 18, 177, 223, 238, 369, 441, 621, 622, 636, 648, 649, 666
__tag_exclude_struct_headfoot_begin:n	640, 681, 682	__tag_insert_bdc_node	395
__tag_exclude_struct_headfoot_end:	656, 683, 684	__tag_insert_bmc_node	366
__tag_fakespace	445	__tag_insert_emc_node	337
__tag_fakespace:	70, 72, 288	__tag_lastpagelabel:	69, 70, 88
__tag_finish_structure:	13, 16, 336, 337	__tag_log	183
__tag_get_data_mc_counter:	9, 9	\l__tag_loglevel_int	104, 132, 169, 171, 172, 201, 220, 248, 251, 256, 259, 262, 263, 264, 275, 368, 375, 382, 389, 409, 417, 424, 432, 439, 470
__tag_get_data_mc_tag:	254, 254, 349, 349	__tag_mark_spaces	450
__tag_get_data_struct_counter:	477, 478	__tag_mc_artifact_begin_marks:n	19, 41, 77, 377

\l__tag_mc_artifact_bool
..... [20](#), [124](#), [178](#), [192](#), [239](#), [373](#)
\l__tag_mc_artifact_type_tl
..... [19](#), [128](#), [132](#), [136](#),
[140](#), [144](#), [148](#), [152](#), [156](#), [347](#), [375](#), [377](#)
__tag_mc_bdc:nn [229](#), [232](#), [264](#), [306](#), [339](#)
__tag_mc_bdc_mcid:n .. [119](#), [234](#), [311](#)
__tag_mc_bdc_mcid:nn
..... [234](#), [237](#), [267](#), [313](#), [318](#)
__tag_mc_bdc_shipout:nn ... [233](#), [245](#)
__tag_mc_begin_marks:nn
..... [19](#), [19](#), [40](#), [76](#), [384](#)
__tag_mc_bmc:n [229](#), [230](#), [335](#)
__tag_mc_bmc_artifact: [333](#), [333](#), [346](#)
__tag_mc_bmc_artifact:n [333](#), [337](#), [347](#)
\l__tag_mc_botmarks_seq
..... [81](#), [17](#), [86](#), [107](#),
[157](#), [187](#), [204](#), [204](#), [212](#), [217](#), [340](#), [357](#)
__tag_mc_disable_marks: [74](#), [74](#)
__tag_mc_emc: [154](#), [229](#), [231](#), [443](#)
__tag_mc_end_marks: . [19](#), [59](#), [78](#), [444](#)
\l__tag_mc_firstmarks_seq
..... [80](#), [17](#), [83](#), [106](#), [186](#), [192](#),
[195](#), [196](#), [203](#), [203](#), [204](#), [340](#), [348](#), [350](#)
\g__tag_mc_footnote_marks_seq ... [14](#)
__tag_mc_get_marks: [80](#), [80](#), [178](#), [199](#)
__tag_mc_handle_artifact:N
..... [115](#), [333](#), [341](#), [375](#)
__tag_mc_handle_mc_label:n
..... [26](#), [26](#), [197](#), [388](#)
__tag_mc_handle_mcid:nn
..... [234](#), [316](#), [321](#), [381](#)
__tag_mc_handle_stash:n [49](#), [138](#),
[140](#), [141](#), [170](#), [227](#), [322](#), [322](#), [332](#), [416](#)
__tag_mc_if_in: [66](#), [80](#), [221](#), [228](#)
__tag_mc_if_in:TF [66](#), [86](#), [221](#), [232](#), [240](#)
__tag_mc_if_in_p: [66](#), [221](#)
__tag_mc_insert_extra_tmb:n ...
..... [104](#), [104](#), [167](#)
__tag_mc_insert_extra_tme:n ...
..... [104](#), [149](#), [168](#)
__tag_mc_insert_mcid_kids:n ...
..... [129](#), [129](#), [148](#), [269](#)
__tag_mc_insert_mcid_single_
kids:n [129](#), [134](#), [270](#)
\l__tag_mc_key_label_tl
. [22](#), [194](#), [197](#), [319](#), [384](#), [385](#), [388](#), [489](#)
\l__tag_mc_key_properties_tl ...
..... [22](#), [179](#), [268](#), [283](#), [284](#),
[304](#), [305](#), [383](#), [463](#), [472](#), [473](#), [485](#), [486](#)
\l__tag_mc_key_stash_bool
..... [20](#), [27](#), [36](#), [123](#), [200](#), [390](#)
\g__tag_mc_key_tag_tl ... [19](#), [22](#),
[182](#), [242](#), [254](#), [260](#), [349](#), [371](#), [442](#), [459](#)
\l__tag_mc_key_tag_tl [22](#), [181](#), [189](#),
[191](#), [241](#), [259](#), [370](#), [380](#), [382](#), [384](#), [458](#)
__tag_mc_lua_set_mc_type_attr:n
..... [81](#), [81](#), [105](#), [191](#)
__tag_mc_lua_unset_mc_type_
attr: [81](#), [107](#), [240](#)
\g__tag_mc_main_marks_seq [14](#)
\g__tag_mc_marks [13](#),
[21](#), [30](#), [43](#), [50](#), [61](#), [67](#), [84](#), [87](#), [193](#), [213](#)
\g__tag_mc_multicol_marks_seq ... [14](#)
\g__tag_mc_parenttree_prop
..... [17](#), [18](#), [99](#), [164](#), [182](#), [328](#)
\l__tag_mc_ref_abspage_tl
..... [11](#), [270](#), [282](#), [290](#), [298](#)
__tag_mc_set_label_used:n [30](#), [30](#), [50](#)
\g__tag_mc_stack_seq
..... [18](#), [88](#), [95](#), [105](#), [254](#)
__tag_mc_store:nnn . [89](#), [89](#), [103](#), [130](#)
\l__tag_mc_tmpa_tl .. [12](#), [284](#), [287](#), [291](#)
g__tag_MCID_abs_int [7](#)
\g__tag_MCID_byabspage_prop
..... [262](#), [280](#), [289](#), [297](#)
\g__tag_MCID_tmp_bypage_int
..... [263](#), [266](#), [287](#), [295](#), [308](#)
\g__tag_mode_lua_bool
... [41](#), [49](#), [50](#), [114](#), [238](#), [278](#), [289](#),
[298](#), [304](#), [368](#), [560](#), [616](#), [631](#), [643](#), [661](#)
__tag_new_output_prop_handler:n
..... [68](#), [78](#), [102](#), [764](#)
__tag_pairs_prop [200](#)
\l__tag_para_attr_class_tl
..... [292](#), [382](#), [484](#)
\g__tag_para_begin_int
..... [292](#), [316](#), [334](#), [421](#), [542](#), [547](#)
\l__tag_para_bool [292](#), [388](#), [397](#), [404](#),
[410](#), [441](#), [460](#), [493](#), [592](#), [593](#), [615](#), [642](#)
\g__tag_para_end_int
..... [292](#), [324](#), [342](#), [432](#), [542](#), [548](#)
\l__tag_para_flattened_bool
..... [292](#), [393](#), [400](#), [413](#), [443](#), [468](#), [500](#)
\l__tag_para_main_attr_class_tl .
..... [292](#), [474](#)
\g__tag_para_main_begin_int
..... [292](#), [312](#), [330](#), [533](#), [538](#)
\g__tag_para_main_end_int
..... [292](#), [320](#), [338](#), [533](#), [539](#)
__tag_para_main_store_struct: ..
..... [352](#), [352](#), [450](#), [476](#)
\g__tag_para_main_struct_tl [292](#), [354](#)
\l__tag_para_main_tag_tl
..... [292](#), [392](#), [399](#), [412](#), [448](#), [473](#)
\l__tag_para_show_bool
..... [292](#), [389](#), [390](#), [405](#), [418](#), [429](#)
\l__tag_para_tag_default_tl ... [292](#)

`\l__tag_para_tag_tl`
 292, 360, 391, 398, 406, 411, 453, 483
`\l__tag_parent_child_check_tl` ...
 210, 211, 400, 401, 447,
 671, 672, 853, 854, 998, 999, 1061, 1062
`__tag_parenttree_add_objr:nm` ...
 159, 159, 447
`\l__tag_parenttree_content_tl` ...
 166, 185, 197, 217, 225, 246, 249
`\g__tag_parenttree_objr_tl`
 158, 161, 246
`__tag_pdf_name_e:n` 98, 98
`__tag_pdf_object_ref` 425
`__tag_prop_gput:Nnn`
 9, 29, 90, 119, 126,
 130, 165, 168, 175, 288, 296, 303, 976
`__tag_prop_item:Nn` .. 9, 49, 165, 171
`__tag_prop_new:N` 9,
 9, 11, 101, 165, 165, 177, 262, 762
`__tag_prop_new_linked:N`
 15, 17, 165, 166
`__tag_prop_show:N` 9, 62, 165, 173, 180
`__tag_property_gset:nmmn`
 127, 128, 265
`\c__tag_property_mc_clist`
 102, 244, 305
`__tag_property_new:nmmn`
 127, 127, 145, 148, 155, 158, 162
`__tag_property_record:nm`
 28, 131, 140, 240, 301, 434, 784
`__tag_property_ref:nm` . 130, 139, 597
`__tag_property_ref:nmm`
 41, 127, 129, 138,
 143, 177, 181, 185, 199, 200, 272,
 317, 328, 443, 968, 974, 977, 983, 990
`__tag_property_ref_lastpage:nm` .
 . 81, 141, 141, 156, 170, 173, 296, 310
`\c__tag_property_struct_clist` ...
 102, 786
`g__tag_role/RoleMap_dict` 18
`\g__tag_role_add_mathml_bool` ...
 73, 255, 690, 740
`__tag_role_add_tag:nm`
 127, 127, 155, 282, 361, 725
`__tag_role_add_tag:nmmn`
 169, 169, 228, 314, 730
`__tag_role_alloctag:nmm` 81,
 85, 95, 107, 117, 126, 142, 186, 279, 310
`\l__tag_role_debug_prop`
 155, 11, 506, 507, 579, 580
`__tag_role_get:nmN`
 156, 158, 166, 229, 231, 255, 498, 798
`__tag_role_get_parent_child_-`
 rule:nmmN 169, 447, 448, 500, 532, 639
`\g__tag_role_index_prop` . 155, 10,
 404, 412, 424, 425, 426, 431, 437,
 439, 440, 443, 445, 452, 453, 508, 518
`\g__tag_role_NS<ns>_class_prop` 155
`\g__tag_role_NS<ns>_prop` 155
`\g__tag_role_NS_mathml_prop` 257, 441
`__tag_role_NS_new:nmm`
 . 157, 20, 22, 30, 74, 75, 76, 77, 78, 80
`\g__tag_role_NS_prop`
 ... 155, 9, 26, 56, 166, 314, 332, 713
`\g__tag_role_parent_child_-`
 intarray 380, 383, 461
`__tag_role_read_namespace:n` 339,
 339, 343, 344, 345, 347, 349, 351, 352
`__tag_role_read_namespace:nm` ...
 320, 320, 341, 350
`__tag_role_read_namespace_-`
 line:nw 257, 261, 294, 330
`__tag_role_remap:`
 678, 678, 679, 871, 1003, 1066
`__tag_role_remap_id:` 679, 679
`\l__tag_role_remap_NS_tl`
 676, 870, 873, 1002, 1005, 1065, 1068
`\l__tag_role_remap_tag_tl`
 676, 869, 872, 1001, 1004, 1064, 1067
`\l__tag_role_role_namespace_-`
 tmpa_tl 12,
 685, 706, 711, 713, 715, 719, 734
`\l__tag_role_role_tmpa_tl`
 12, 684, 704, 710, 727, 733
`\g__tag_role_rolemap_prop`
 155, 18, 145, 148, 151, 160,
 216, 219, 222, 259, 262, 375, 513, 523
`\c__tag_role_rules_num_prop` 381, 472
`\c__tag_role_rules_prop` 381, 384, 465
`\l__tag_role_tag_namespace_tmpa_-`
 tl 12, 555, 559, 563, 683, 732
`\l__tag_role_tag_namespace_tmpb_-`
 tl 14, 556, 557, 560, 564
`\l__tag_role_tag_namespace_tmpb_-`
 tluuuuuu% 12
`\l__tag_role_tag_tmpa_tl`
 12, 682, 703, 726, 731
`\g__tag_role_tags_class_prop` ...
 ... 155, 8, 90, 99, 112, 121, 137, 270
`\g__tag_role_tags_NS_prop` .. 155,
 7, 88, 97, 110, 119, 130, 164, 199,
 263, 373, 487, 495, 555, 556, 709, 945
`\l__tag_role_tmpa_seq` 12
`\l__tag_role_update_bool`
 210, 257, 258, 266, 346, 348
`\c__tag_role_userNS_id_str`
 156, 59, 80
`\g__tag_root_default_tl` 255

\g__tag_saved_in_mc_bool	g__tag_struct_kids_0_seq
. 612, 621, 636, 648, 666 100
__tag_seq_gput_right:Nn 9,	__tag_struct_mcid_dict:n
36, 165, 169, 176, 208, 218, 228, 251 94, 97, 193, 211
__tag_seq_item:Nn 9, 44, 165, 170	\c__tag_struct_null_tl 10, 310
__tag_seq_new:N	\g__tag_struct_objR_seq 8
. 9, 9, 22, 103, 165, 167, 178, 765	__tag_struct_output_prop_aux:nn
__tag_seq_show:N 9, 55, 165, 172, 179 68, 68, 82
__tag_show_spacemark 431	__tag_struct_prop_gput:nnn
\l__tag_showspace_bool 15, 16 86, 87, 88, 94, 105, 110,
__tag_space_chars_shipout 543	115, 120, 127, 153, 162, 168, 312,
__tag_start_para_ints:	325, 339, 536, 548, 562, 578, 586,
. 201, 224, 326, 326	609, 631, 672, 691, 734, 770, 803,
__tag_stop_para_ints:	821, 830, 879, 1039, 1105, 1178, 1229
. 191, 214, 326, 345	\g__tag_struct_ref_by_dest_prop . 67
__tag_store_parent_child_-	\g__tag_struct_roletag_NS_tl . . . 58
rule:nnn 381, 381, 418	\l__tag_struct_roletag_NS_tl . . .
g__tag_struct_0_prop 100 61, 802, 807, 834
__tag_struct_add_AF:nn	\l__tag_struct_roletag_tl
. 628, 645, 664, 671, 690, 733 58, 801, 807, 809, 834, 838
__tag_struct_add_inline_AF:nn . .	__tag_struct_set_tag_info:nnn . .
. 617, 644, 704, 708, 715, 723	148, 150, 160, 175, 776, 874, 1006, 1069
\g__tag_struct_AFobj_int 615, 623, 626	\g__tag_struct_stack_current_tl .
\g__tag_struct_cont_mc_prop 16, 25, 34, 65, 71, 97, 146,
. 11, 91, 92, 94, 97, 221	152, 160, 166, 203, 214, 224, 280,
\g__tag_struct_dest_num_prop . . . 64	326, 330, 393, 404, 413, 463, 468,
\l__tag_struct_elem_stash_bool . .	474, 839, 886, 890, 891, 912, 930,
. 63, 507, 842, 901	936, 973, 980, 986, 1036, 1043, 1049
__tag_struct_exchange_kid_-	\l__tag_struct_stack_parent_-
command:N 265, 265, 275, 306	tmpa_tl 16, 428, 437,
__tag_struct_fill_kid_key:n	452, 517, 774, 788, 792, 817, 845,
. 134, 276, 276, 404	857, 866, 883, 887, 889, 892, 904, 913
__tag_struct_format_Ref:n	\g__tag_struct_stack_seq 12, 22, 25,
. 385, 385, 386	427, 662, 791, 797, 840, 923, 928, 934
__tag_struct_get_dict_content:nN	\c__tag_struct_StructElem_-
. 135, 356, 356, 405	entries_seq 21
__tag_struct_get_id:n	\c__tag_struct_StructTreeRoot_-
. 94, 99, 112, 113, 137, 138, 412, 468	entries_seq 21
__tag_struct_get_parentrole:nNN	\g__tag_struct_tag_NS_tl 58, 489,
. 176,	497, 498, 500, 779, 800, 852, 864,
176, 192, 202, 392, 663, 844, 989, 1052	870, 873, 877, 911, 947, 995, 1002,
__tag_struct_gput_data_ref:nn . .	1005, 1009, 1058, 1065, 1068, 1072
. 599, 1097, 1098, 1110	\g__tag_struct_tag_stack_seq . . .
__tag_struct_insert_annot:nn 14, 45,
. 420, 420, 1116	219, 220, 412, 427, 441, 837, 927, 941
\l__tag_struct_key_label_tl	\g__tag_struct_tag_tl
. 62, 506, 782, 785 58, 181, 182, 185, 370,
__tag_struct_kid_mc_gput_-	371, 488, 490, 496, 498, 499, 501,
right:nn . . . 193, 205, 206, 224, 325	778, 799, 838, 851, 864, 869, 872,
__tag_struct_kid_OBJR_gput_-	876, 943, 945, 987, 994, 1001, 1004,
right:nnn . . 241, 241, 244, 264, 435	1008, 1050, 1057, 1064, 1067, 1071
__tag_struct_kid_struct_gput_-	__tag_struct_write_obj:n
right:nn 147, 387, 387
. 225, 225, 226, 240, 888, 972, 1035	\l__tag_tag_stop_int 182, 186, 187,
	195, 196, 203, 209, 210, 218, 219, 226

<code>\g__tag_tagunmarked_bool</code>	116 , 268 , 270	<code>\g__tag_tree_openaction_struct_</code>	
<code>\l__tag_tmpa_box</code>	<code>tl</code>	30 , 36 , 55
	89 , 167 , 173 , 174 , 178 , 189 , 190	<code>__tag_tree_parenttree_rerun_</code>	
<code>\l__tag_tmpa_clist</code>	<code>msg:</code>	167 , 210 , 245
	89 , 1156 , 1157 , 1190 , 1191 , 1193	<code>__tag_tree_update_openaction:</code>	..
<code>\l__tag_tmpa_int</code>		40 , 73
	88 , 89 , 91 , 96 , 99 , 103 , 112 , 386 , 398 , 400 , 470	<code>__tag_tree_write_classmap:</code>
<code>\l__tag_tmpa_prop</code>	89 , 172 , 180 , 193 , 195		276 , 276 , 346
<code>\l__tag_tmpa_seq</code>	<code>__tag_tree_write_idtree:</code>	... 84 , 344
	49 , 56 , 57 , 89 , 280 , 280 , 282 , 284 , 285 , 286 , 287 , 292 , 399 , 402 , 410 , 411 , 413 , 414 , 415 , 487 , 488 , 489 , 495 , 496 , 497 , 1158 , 1162 , 1172 , 1173 , 1174 , 1176 , 1194 , 1200 , 1202 , 1226	<code>__tag_tree_write_namespaces:</code>	...
<code>\l__tag_tmpa_str</code>		310 , 310 , 347
	42 , 43 , 48 , 94 , 279 , 284 , 289 , 300 , 305 , 312 , 468 , 473 , 481 , 486 , 532 , 539 , 544 , 551 , 558 , 565 , 574 , 581 , 605 , 612	<code>__tag_tree_write_parenttree:</code>	...
<code>\l__tag_tmpa_tl</code>		236 , 236 , 343
	41 , 42 , 45 , 47 , 49 , 49 , 54 , 84 , 86 , 89 , 91 , 92 , 92 , 94 , 100 , 104 , 105 , 107 , 107 , 112 , 113 , 114 , 114 , 115 , 135 , 137 , 138 , 140 , 140 , 142 , 145 , 146 , 151 , 179 , 180 , 182 , 185 , 186 , 187 , 195 , 196 , 199 , 200 , 201 , 201 , 207 , 216 , 216 , 217 , 222 , 224 , 267 , 270 , 271 , 271 , 272 , 273 , 277 , 278 , 279 , 289 , 294 , 296 , 301 , 302 , 302 , 302 , 304 , 308 , 308 , 310 , 310 , 405 , 406 , 411 , 411 , 412 , 413 , 413 , 414 , 424 , 425 , 426 , 431 , 437 , 439 , 441 , 443 , 445 , 449 , 452 , 455 , 462 , 472 , 474 , 483 , 498 , 499 , 508 , 510 , 513 , 515 , 529 , 533 , 583 , 591 , 593 , 593 , 594 , 596 , 596 , 600 , 600 , 605 , 627 , 630 , 636 , 640 , 662 , 664 , 859 , 866 , 927 , 928 , 934 , 936 , 941 , 944 , 945 , 947 , 991 , 996 , 1030 , 1054 , 1059 , 1170 , 1181	<code>__tag_tree_write_rolemap:</code>
<code>\l__tag_tmpb_box</code>		253 , 253 , 345
	89 , 168 , 175 , 176 , 180 , 182	<code>__tag_tree_write_structelements:</code>
<code>\l__tag_tmpb_seq</code>		143 , 143 , 348
	89 , 1157 , 1158 , 1193 , 1194	<code>__tag_tree_write_structtreeroot:</code>
<code>\l__tag_tmpb_tl</code>		124 , 124 , 349
	.. 167 , 87 , 89 , 102 , 116 , 118 , 391 , 412 , 418 , 440 , 445 , 453 , 456 , 462 , 476 , 498 , 500 , 518 , 520 , 523 , 525 , 530 , 533 , 610 , 618 , 620 , 621 , 623 , 627 , 632 , 637 , 640 , 992 , 997 , 1055 , 1060	<code>__tag_whatsits:</code>	35 , 61 , 62 , 65 , 351 , 352
<code>__tag_tree_fill_parenttree:</code>	...	<code>tag-namespace_(rolemap-key)</code> 680
	167 , 168 , 243	<code>tag/struct/0</code> internal commands:	
<code>__tag_tree_final_checks:</code>	20 , 20 , 342	<code>__tag/struct/0</code> 29
<code>\g__tag_tree_id_pad_int</code>	.. 76 , 80 , 143	<code>tag/tree/namespaces</code> internal commands:	
<code>__tag_tree_lua_fill_parenttree:</code>	<code>__tag/tree/namespaces</code> 309
	223 , 223 , 240	<code>tag/tree/parenttree</code> internal commands:	
		<code>__tag/tree/parenttree</code> 150
		<code>tag/tree/rolemap</code> internal commands:	
		<code>__tag/tree/rolemap</code> 252
		<code>tagabspage</code> 6 , 145
		<code>tagmcabs</code> 6 , 145
		<code>\tagmcbegin</code> 35 , 155 , 22 , 369 , 375
		<code>\tagmccend</code> 35 , 22 , 375
		<code>tagmccid</code> 6 , 145
		<code>\tagmccifin</code> 35
		<code>\tagmccifinTF</code> 35 , 39
		<code>\tagmccuse</code> 35 , 22
		<code>\tagpdfparaOff</code> 37 , 589
		<code>\tagpdfparaOn</code> 37 , 589
		<code>\tagpdfsetup</code> 35 , 101 , 154 , 6
		<code>\tagpdfsuppressmarks</code> 37 , 594
		<code>\tagstart</code> 6 , 206 , 233
		<code>\tagstop</code> 6 , 205 , 232
		<code>tagstruct</code> 6 , 145
		<code>\tagstructbegin</code>
			36 , 154 , 155 , 45 , 258 , 360 , 362
		<code>\tagstructend</code> 36 , 45 , 259 , 375
		<code>tagstructobj</code> 6 , 145
		<code>\tagstructuse</code> 36 , 45
		<code>\tagtool</code> 35 , 13
		<code>tagunmarked_(deprecated)</code> 6 , 268
		<code>TeX and L^AT_EX 2_ε commands:</code>	
		<code>\@M</code> 164
		<code>\@auxout</code> 74

<code>\@bsphack</code>	133	268, 354, 442, 459, 488, 489, 496,
<code>\@cclv</code>	569	497, 499, 500, 659, 839, 936, 943, 947
<code>\@esphack</code>	135	<code>\tl_gset_eq:NN</code>
<code>\@gobble</code>	31, 55	182, 371
<code>\@ifpackageloaded</code>	28, 552	<code>\tl_head:N</code>
<code>\@kernel@after@foot</code>	608	593, 620
<code>\@kernel@after@head</code>	606	<code>\tl_if_empty:NTF</code>
<code>\@kernel@before@cclv</code>	559, 566	42, 43, 107, 194, 259,
<code>\@kernel@before@foot</code>	607	295, 345, 385, 594, 621, 700, 706, 781
<code>\@kernel@before@footins</code>	562, 564	<code>\tl_if_empty:nTF</code>
<code>\@kernel@before@head</code>	603, 605	51, 55,
<code>\@kernel@tag@hangfrom</code>	358	63, 75, 143, 194, 198, 212, 213, 264,
<code>\@kernel@tagsupport@makecol</code>	558, 571	268, 276, 297, 297, 299, 396, 460,
<code>\@makecol</code>	568, 573	468, 478, 555, 571, 581, 608, 620, 688
<code>\@maxdepth</code>	177	<code>\tl_if_empty_p:n</code>
<code>\@mult@ptagging@hook</code>	576	312
<code>\@outputbox</code>	574	<code>\tl_if_eq:NNTF</code>
<code>\@secondoftwo</code>	31, 55	310, 340
<code>\@tempboxa</code>	363, 373, 375	<code>\tl_if_eq:NnTF</code>
<code>\c@page</code>	568, 573	107
<code>\count@</code>	581	<code>\tl_if_eq:nnTF</code>
<code>\mult@firstbox</code>	579	214, 264, 280
<code>\mult@rightbox</code>	583	<code>\tl_if_exist:NTF</code> ...
<code>\new@label@record</code>	76	121, 307, 380, 647
<code>\on@line</code>	465, 480, 496	<code>\tl_if_head_eq_charcode:nNTF</code> ...
<code>\page@sofar</code>	578	47
<code>\process@cols</code>	579	<code>\tl_if_in:nnTF</code>
tex commands:		187
<code>\tex_botmarks:D</code>	87	<code>\tl_new:N</code>
<code>\tex_firstmarks:D</code>	84	11, 12, 12, 13,
<code>\tex_kern:D</code>	180	14, 15, 16, 17, 19, 20, 22, 23, 24, 25,
<code>\tex_marks:D</code>	21, 30, 43, 50, 61, 67	30, 32, 58, 59, 60, 61, 62, 89, 90, 91,
<code>\tex_special:D</code>	65	92, 93, 158, 166, 255, 300, 301, 303,
<code>\tex_splitbotmarks:D</code>	213	305, 308, 309, 447, 657, 676, 677, 1132
<code>\tex_splitfirstmarks:D</code>	193	<code>\tl_put_left:Nn</code>
texsource	100	606, 608
<code>\the</code>	568, 573	<code>\tl_put_right:Nn</code>
<code>\tiny</code>	421, 432	92, 102, 116, 185, 197,
<code>title_␣(struct-key)</code>	99, 504	216, 246, 268, 283, 284, 304, 305,
<code>title-o_␣(struct-key)</code>	99, 504	366, 463, 472, 473, 485, 486, 564,
tl commands:		566, 571, 576, 596, 605, 607, 1217, 1224
<code>\c_empty_tl</code>	361, 375	<code>\tl_set:Nn</code>
<code>\c_space_tl</code> 53, 54, 74, 102, 163, 187,		41,
188, 193, 196, 198, 200, 206, 249,		84, 114, 128, 132, 136, 140, 140,
282, 349, 373, 411, 568, 573, 654,		144, 148, 152, 156, 162, 164, 182,
867, 904, 986, 1049, 1108, 1173, 1219		184, 185, 185, 225, 240, 241, 241,
<code>\tl_clear:N</code>	86, 87,	245, 246, 251, 252, 259, 270, 273,
104, 179, 188, 189, 278, 358, 557, 593		277, 279, 289, 294, 302, 304, 304,
<code>\tl_const:Nn</code>	10	306, 308, 308, 319, 382, 458, 458,
<code>\tl_count:n</code>	77, 81, 143	474, 476, 491, 510, 515, 517, 520,
<code>\tl_gput_right:Nn</code>	161, 652	525, 538, 568, 583, 593, 596, 600,
<code>\tl_gset:Nn</code>		605, 610, 620, 623, 627, 632, 645,
.... 18, 31, 36, 97, 242, 256, 260,		703, 704, 715, 719, 774, 1170, 1198
		<code>\tl_set_eq:NN</code>
		181, 370
		<code>\tl_show:N</code>
		886, 887, 1222, 1228
		<code>\tl_tail:n</code>
		461
		<code>\tl_to_str:n</code>
	 32, 47, 148, 204, 219, 363, 396
		<code>\tl_trim_spaces:n</code>
		47
		<code>\tl_use:N</code>
		123, 636, 677, 696, 739
		<code>\l_tmpa_tl</code>
		199, 218, 699, 700, 702
token commands:		
<code>\token_to_str:N</code>	76, 568, 573	
tree-mcid-index-wrong	20, 85	
tree-struct-still-open	20, 42	

U	
<code>uncompress_□</code> (deprecated)	255
<code>unittag_□</code> (deprecated)	386
<code>\unskip</code>	35
use commands:	
<code>\use:N</code>	94 , 595
<code>\use:n</code>	41 , 309
<code>\use_i:nn</code>	
.	184 , 240 , 361 , 375 , 445 , 449 , 944
<code>\use_ii:nn</code>	185 , 241 , 332
<code>\use_none:n</code>	77 , 86 , 89
<code>\use_none:nn</code>	76 , 87 , 1091
<code>\UseSocket</code>	38 , 39 , 70 , 75 , 80
<code>\UseTaggingSocket</code>	38 , 39 , 64 , 66
V	
<code>\vbadness</code>	164 , 188
vbox commands:	
<code>\vbox_set_split_to_ht:NNn</code>	190
<code>\vbox_set_to_ht:Nnn</code>	166
<code>\vbox_unpack_drop:N</code>	179
<code>\vfuzz</code>	165
<code>viewer/startpage_□</code> (setup-key)	32
W	
<code>\wd</code>	373