

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

Ulrike Fischer[†]

Released 2023-12-18

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	16
1	Commands	16

^{*}This file describes v0.98r, last revised 2023-12-18.

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

2	Description of log messages	16
2.1	\ShowTagging command	16
2.2	Messages in checks and commands	17
2.3	Messages from the ptagging code	17
2.4	Warning messages from the lua-code	17
2.5	Info messages from the lua-code	17
2.6	Debug mode messages and code	18
2.7	Messages	18
3	Messages	20
3.1	Messages related to mc-chunks	20
3.2	Messages related to structures	21
3.3	Attributes	22
3.4	Roles	22
3.5	Miscellaneous	23
4	Retrieving data	24
5	User conditionals	24
6	Internal checks	25
6.1	checks for active tagging	25
6.2	Checks related to structures	25
6.3	Checks related to roles	27
6.4	Check related to mc-chunks	27
6.5	Checks related to the state of MC on a page or in a split stream	30
II The tagpdf-user module		
Code related to L ^A T _E X2e user commands and document commands		
Part of the tagpdf package		34
1	Setup commands	34
2	Commands related to mc-chunks	34
3	Commands related to structures	35
4	Debugging	35
5	Extension commands	36
5.1	Fake space	36
5.2	Paratagging	36
5.3	Header and footer	37
5.4	Link tagging	37
6	Socket support	37
7	User commands and extensions of document commands	38
8	Setup and preamble commands	38

9	Commands for the mc-chunks	38
10	Commands for the structure	39
11	Socket support	40
12	Debugging	40
13	Commands to extend document commands	44
13.1	Document structure	44
13.2	Structure destinations	45
13.3	Fake space	45
13.4	Paratagging	45
13.5	Header and footer	51
13.6	Links	53

III The **tagpdf-tree** module

Commands trees and main dictionaries

Part of the tagpdf package **55**

1	Trees, pdfmanagement and finalization code	55
1.1	Check structure	55
1.2	Catalog: MarkInfo and StructTreeRoot	56
1.3	Writing the IDtree	56
1.4	Writing structure elements	57
1.5	ParentTree	58
1.6	Rolemap dictionary	61
1.7	Classmap dictionary	62
1.8	Namespaces	62
1.9	Finishing the structure	63
1.10	StructParents entry for Page	64

IV The **tagpdf-mc-shared** module

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

Part of the tagpdf package **65**

1	Public Commands	65
2	Public keys	66
3	Marked content code – shared	67
3.1	Variables and counters	67
3.2	Functions	68
3.3	Keys	71

V	The <code>tagpdf-mc-generic</code> module	
	Code related to Marked Content (<code>mc-chunks</code>), generic mode	
	Part of the <code>tagpdf</code> package	72
1	Marked content code – generic mode	72
1.1	Variables	72
1.2	Functions	73
1.3	Looking at MC marks in boxes	76
1.4	Keys	84
VI	The <code>tagpdf-mc-luacode</code> module	
	Code related to Marked Content (<code>mc-chunks</code>), luamode-specific	
	Part of the <code>tagpdf</code> package	86
1	Marked content code – luamode code	86
1.1	Commands	88
1.2	Key definitions	92
VII	The <code>tagpdf-struct</code> module	
	Commands to create the structure	
	Part of the <code>tagpdf</code> package	95
1	Public Commands	95
2	Public keys	96
2.1	Keys for the structure commands	96
2.2	Setup keys	98
3	Variables	98
3.1	Variables used by the keys	100
3.2	Variables used by tagging code of basic elements	101
4	Commands	101
4.1	Initialization of the <code>StructTreeRoot</code>	102
4.2	Adding the <code>/ID</code> key	103
4.3	Filling in the tag info	103
4.4	Handlings kids	104
4.5	Output of the object	108
5	Keys	112
6	User commands	117
7	Attributes and attribute classes	125
7.1	Variables	125
7.2	Commands and keys	125

VIII	The <code>tagpdf-luatex.def</code>	
	Driver for luatex	
	Part of the tagpdf package	129
1	Loading the lua	129
2	Logging functions	133
3	Helper functions	135
3.1	Retrieve data functions	135
3.2	Functions to insert the pdf literals	137
4	Function for the real space chars	139
5	Function for the tagging	143
6	Parenttree	148
IX	The <code>tagpdf-roles</code> module	
	Tags, roles and namespace code	
	Part of the tagpdf package	150
1	Code related to roles and structure names	150
1.1	Variables	151
1.2	Namespaces	153
1.3	Adding a new tag	154
1.3.1	pdf 1.7 and earlier	155
1.3.2	The pdf 2.0 version	157
1.4	Helper command to read the data from files	158
1.5	Reading the default data	160
1.6	Parent-child rules	161
1.6.1	Reading in the csv-files	161
1.6.2	Retrieving the parent-child rule	163
1.7	Remapping of tags	168
1.8	Key-val user interface	169
X	The <code>tagpdf-space</code> module	
	Code related to real space chars	
	Part of the tagpdf package	171
1	Code for interword spaces	171
	Index	174

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

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

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

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

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

Keys to activate the various tagging steps

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

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

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

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

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

1 Initialization and test if pdfmanagement is active.

```

1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2023-12-18} {0.98r}
4 { A package to experiment with pdf tagging }
5
6 \bool_if:nF
7 {
8   \bool_lazy_and_p:nn
9     {\cs_if_exist_p:N \pdfmanagement_if_active_p:}
10    { \pdfmanagement_if_active_p: }
11 }
12 { %error for now, perhaps warning later.
13   \PackageError{tagpdf}
14     {
15       PDF~resource~management~is~no~active!\MessageBreak
16       tagpdf~will~no~work.
17     }
18     {
19       Activate~it~with \MessageBreak
20       \string\RequirePackage{pdfmanagement-testphase}\MessageBreak
21       \string\DocumentMetadata{<options>}\MessageBreak
22       before~\string\documentclass
23     }
24 }
25 </package>
26
27 <*debug>
28 \ProvidesExplPackage {tagpdf-debug} {2023-12-18} {0.98r}
29 { debug code for tagpdf }
30 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}}\endinput
31 </debug> We map the internal module name “tag” to “tagpdf” in messages.
32 <*package>
33 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
34 </package>
35
36 Debug mode has its special mapping:
37 <*debug>
38 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug } {}
39 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
40 </debug>

```

2 base package

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

```

36 <*base>
37 \ProvidesExplPackage {tagpdf-base} {2023-12-18} {0.98r}
38 {part of tagpdf - provide base, no-op versions of the user commands }
39 </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:NnF \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 { @files }
73   {
```



```

74     \exp_args:NNe \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 if pdf version 2.0 is detected, but with the boolean struct-dest-boolean one can suppress them. Also we assume currently that they are set only at begin document. But if some control passing over groups are needed they could be perhaps used in a document too. TODO: check if they are used everywhere as needed and as wanted.

```

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}
122 \cs_generate_variant:Nn \prop_put:Nnn {Nee}
123 \cs_generate_variant:Nn \prop_item:Nn {No,Ne}
124 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
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.

```

127 \_tag_property_new:nnnn
128 \_tag_property_gset:nnnn
129 \_tag_property_ref:nnn
130
131 \cs_new_eq:NN \_tag_property_new:nnnn \property_new:nnnn
132
133 \cs_new_eq:NN \_tag_property_gset:nnnn \property_gset:nnnn
134
135 \cs_new_eq:NN \_tag_property_ref:nnn \property_ref:nnn
136
137 \cs_new_eq:NN \_tag_property_ref:nn \property_ref:nn
138
139 \cs_new_protected:Npn \_tag_property_record:nn #1#2
140 {
141   \@bsphack
142   \property_record:nn{#1}{#2}
143   \@esphack
144 }

```

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 **tagabspace**, the absolute id **tagmcabc**, 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:nnnn
156 { tagabspage } { shipout }
157 {0} { \int_use:N \g_shipout_readonly_int }
158 \__tag_property_new:nnnn { tagmcabs } { now }
159 {0} { \int_use:N \c@g__tag_MCID_abs_int }
160
161 \flag_new:n { __tag/mcid }
162 \__tag_property_new:nnnn {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_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
165 \cs_set_eq:NN \__tag_prop_new:N \prop_new:N
166 \cs_set_eq:NN \__tag_seq_new:N \seq_new:N
167 \cs_set_eq:NN \__tag_prop_gput:Nnn \prop_gput:Nnn
168 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
169 \cs_set_eq:NN \__tag_seq_item:cn \seq_item:cn
170 \cs_set_eq:NN \__tag_prop_item:cn \prop_item:cn
171 \cs_set_eq:NN \__tag_seq_show:N \seq_show:N
172 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
173 % cnx temporary needed for latex-lab-graphic code
174 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nen , Nee, Nne , cnn, cen, cne, cno, cnx}
175 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Ne , No, cn, ce }
176 \cs_generate_variant:Nn \__tag_prop_new:N { c }
177 \cs_generate_variant:Nn \__tag_seq_new:N { c }
178 \cs_generate_variant:Nn \__tag_seq_show:N { c }
179 \cs_generate_variant:Nn \__tag_prop_show:N { c }
180 \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 181 <*package | debug>
182 <package>\int_new:N \l__tag_tag_stop_int

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

```

```

230 \cs_new_protected:Npn \tag_start:{}
231 \cs_new_protected:Npn \tagstop{}
232 \cs_new_protected:Npn \tagstart{}
233 \cs_new_protected:Npn \tag_stop:n #1 {}
234 \cs_new_protected:Npn \tag_start:n #1 {}
235 </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-space␣(setup-key)` Keys to (globally) activate tagging. `activate-space` activates the additional parsing needed for interword spaces. It is not documented, the parsing is currently implicitly activated by the known key `interwordspace`, as the code will perhaps move to some other place, now that it is better separated. `no-struct-dest` allows to suppress structure destinations.

```

236 <*package>
237 \keys_define:nn { __tag / setup }
238 {
239   activate-space .bool_gset:N = \g__tag_active_space_bool,
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   no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,
247

```

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

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

```

248   log           .choice:,
249   log / none    .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
250   log / v       .code:n =
251     {
252       \int_set:Nn \l__tag_loglevel_int { 1 }
253       \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
254     },
255   log / vv      .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
256   log / vvv     .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
257   log / all     .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},

```

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

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

```

258   tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
259   tagunmarked .initial:n = true,

```

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

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

```

260     tabsorder      .choice:,
261     tabsorder / row      .code:n =
262       \pdfmanagement_add:nnn { Page } {Tabs}{/R},
263     tabsorder / column   .code:n =
264       \pdfmanagement_add:nnn { Page } {Tabs}{/C},
265     tabsorder / structure .code:n =
266       \pdfmanagement_add:nnn { Page } {Tabs}{/S},
267     tabsorder / none     .code:n =
268       \pdfmanagement_remove:nn {Page} {Tabs},
269     tabsorder      .initial:n = structure,
270     uncompress      .code:n = { \pdf_uncompress: },
271   }

```

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

12 loading of engine/more dependent code

```

272 \sys_if_engine luatex:T
273 {
274   \file_input:n {tagpdf-luatex.def}
275 }
276 </package>
277 <*mcloading>
278 \bool_if:NTF \g__tag_mode_lua_bool
279 {
280   \RequirePackage {tagpdf-mc-code-lua}
281 }
282 {
283   \RequirePackage {tagpdf-mc-code-generic} %
284 }
285 </mcloading>
286 <*debug>
287 \bool_if:NTF \g__tag_mode_lua_bool
288 {
289   \RequirePackage {tagpdf-debug-lua}
290 }
291 {
292   \RequirePackage {tagpdf-debug-generic} %
293 }
294 </debug>

```

Part I

The tagpdf-checks module

Messages and check code

Part of the tagpdf package

1 Commands

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

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

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

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

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

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

2.7 Messages

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

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

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

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

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

<u>attr-unknown</u>	Message if an attribute is unknown.
---------------------	-------------------------------------

<u>role-missing</u>	Messages related to role mapping.
<u>role-unknown</u>	
<u>role-unknown-tag</u>	
<u>role-tag</u>	
<u>new-tag</u>	
<u>role-parent-child</u>	
<u>role-remapping</u>	

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

<u>sys-no-interwordspace</u>	Message if an engine doesn't support inter word spaces
------------------------------	--

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

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2023-12-18} {0.98r}
4 {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 </header>

```

3 Messages

3.1 Messages related to mc-chunks

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

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

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

mc-tag-missing If the tag is missing

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

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

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

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

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

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

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

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

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

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

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

mc-pushed Informational messages about mc-pushing.

mc-popped

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

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

mc-current Informational messages about current mc state.

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

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

3.2 Messages related to structures

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

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

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

struct-no-objnum Should not happen ...

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

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

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

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

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

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

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

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

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

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

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

struct-used-twice

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

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

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

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

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

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

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

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

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

```

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

```

(End of definition for tree-struct-still-open. This function is documented on page 19.)

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 19.)

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 19.)

3.4 Roles

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

role-unknown

role-unknown-tag

```

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 }

```

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

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

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

(End of definition for `role-parent-child`. This function is documented on page 19.)

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

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

(End of definition for `role-remapping`. This function is documented on page 19.)

role-tag Info messages.

```
new-tag 79 \msg_new:nnn { tag } {role-tag} { mapping-tag~#1~to~role~#2 }
80 \msg_new:nnn { tag } {new-tag} { adding~new~tag~#1 }
81 \msg_new:nnn { tag } {read-namespace} { reading~namespace~definitions~tagpdf~ns~#1~def }
82 \msg_new:nnn { tag } {namespace-missing}{ namespace~definitions~tagpdf~ns~#1~def~not~found }
83 \msg_new:nnn { tag } {namespace-unknown}{ namespace~#1~is~not~declared }
```

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

3.5 Miscellaneous

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

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

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

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

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

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

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

```
88 \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.

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

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

4 Retrieving data

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

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

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

5 User conditionals

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

\tag_if_active:TF

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

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

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

\tag_if_box_tagged:NTF

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



```

126     {
127       \prg_return_false:
128       % warning??
129     }
130   }
131 </base>

```

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

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

```

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

```

(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.

```

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

```

(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.

```

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

```

(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.

```

168 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
169   {
170     \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
171     {
172       \msg_info:nnn { tag } {struct-show-closing} {#1}
173     }
174   }
175
176 \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.

```

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

```

(End of definition for `__tag_check_no_open_struct:`.)

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

```

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

```

(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.

```

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

```

Similar with a namespace

```

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

```

(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).

```

229 \cs_new_protected:Npn \__tag_check_mc_if_nested:
230 {
231   \__tag_mc_if_in:T
232   {

```

```

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

```

(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.

```

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

```

(End of definition for __tag_check_mc_pushed_popped:nn.)

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

```

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

```

(End of definition for __tag_check_mc_tag:N.)

`\g__tag_check_mc_used_intarray` This variable holds the list of used mc numbers. Everytime we store a mc-number we will add one the relevant array index. If everything is right at the end there should be only 1 until the max count of the mcid. 2 indicates that one mcid was used twice, 0 that we lost one. In engines other than luatex the total number of all intarray entries are restricted so we use only a rather small value of 65536, and we initialize the array only at first used, guarded by the log-level. This check is probably only needed for debugging. `__tag_check_init_mc_used:` TODO does this really make sense to check? When can it happen??

```

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

(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.

```

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

(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.

```

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

```

```

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

```

(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.

```

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

```

(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.

`__tag_check_if_mc_tmb_missing:TF`

```

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

```

```

346     {
347         \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
348         ||
349         \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
350     }
351     { \prg_return_true: }
352     { \prg_return_false: }
353 }

```

(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.

```

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

```

(End of definition for __tag_check_if_mc_tme_missing:TF.)

```

360 </package>

```

```

361 <*debug>

```

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

```

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

```

```

389     {
390         \msg_note:nnn { tag / debug } {mc-end } {ignored}
391     }
392 }

```

And now something for the structures

```

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

```



```

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

```

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.

```

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

```

Part II

The tagpdf-user module

Code related to L^AT_EX2e user commands and document commands

Part of the tagpdf package

1 Setup commands

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

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

<code>activate_␣(setup-key)</code>	And additional setup key which combine the other activate keys <code>activate-mc</code> , <code>activate-tree</code> , <code>activate-struct</code> and additionally add a document structure.
------------------------------------	--

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

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

2 Commands related to mc-chunks

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

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

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

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

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

3 Commands related to structures

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

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

4 Debugging

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

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

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

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

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

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

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

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

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

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

<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 Paratagging

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

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

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

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

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

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

5.3 Header and footer

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

```
exclude-header-footer□(setup-key) 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 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.,

$$\text{\UseTaggingSocket\{foo\}} \rightarrow \text{\UseSocket\{tagsupport/foo\}}$$

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} {2023-12-18} {0.98r}
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 34.)

`\tag_tool:n`
`\tagtool`

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

```

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

```

(End of definition for `\tag_tool:n` and `\tagtool`. These functions are documented on page 34.)

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 \tagmcend { }

```

```

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

```

(End of definition for `\tagmcbegin`, `\tagmcend`, and `\tagmcuse`. These functions are documented on page 34.)

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

```

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

```

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

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 35.)

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

```

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

(End of definition for \tag_socket_use:n, \tag_socket_use:nn, and \UseTaggingSocket. These functions are documented on page 37.)

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 35.)

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 35.)

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~absent
135                     =\__tag_get_mc_abs_cnt:"
136                     ,0
137                 )
138                 texio.write_nl("")
139             }
140         }
141     }
142 }

```

```

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 35.)

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

```

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 35.)

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 35.)

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 35.)

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 34.)

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 36.)

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.

<code>\l__tag_para_bool</code>	At first some variables.
<code>\l__tag_para_flattened_bool</code>	
<code>\l__tag_para_show_bool</code>	
<code>\g__tag_para_begin_int</code>	
<code>\g__tag_para_end_int</code>	
<code>\g__tag_para_main_begin_int</code>	
<code>\g__tag_para_main_end_int</code>	
<code>\l__tag_para_tag_default_tl</code>	
<code>\l__tag_para_tag_tl</code>	
<code>\l__tag_para_main_tag_tl</code>	

```

292 </package>
293 <base>\bool_new:N \l__tag_para_flattened_bool
294 <base>\bool_new:N \l__tag_para_bool
295 <*package>
296 \int_new:N \g__tag_para_begin_int
297 \int_new:N \g__tag_para_end_int
298 \int_new:N \g__tag_para_main_begin_int
299 \int_new:N \g__tag_para_main_end_int
300 \tl_new:N \l__tag_para_tag_default_tl
301 \tl_set:Nn \l__tag_para_tag_default_tl { text }
302 \tl_new:N \l__tag_para_tag_tl

```

```

303 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
304 \tl_new:N \l__tag_para_main_tag_tl
305 \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}

```

(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: 306 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
\_tag_gincr_para_end_int: 307 {
308   \int_gincr:N \g__tag_para_main_begin_int
309 }
310 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
311 {
312   \int_gincr:N \g__tag_para_begin_int
313 }
314 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
315 {
316   \int_gincr:N \g__tag_para_main_end_int
317 }
318 \cs_new_protected:Npn \__tag_gincr_para_end_int:
319 {
320   \int_gincr:N \g__tag_para_end_int
321 }

```

(End of definition for __tag_gincr_para_main_begin_int: and others.)

_tag_start_para_ints:
_tag_stop_para_ints:

```

322 \cs_new_protected:Npn \__tag_start_para_ints:
323 {
324   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
325   {
326     \int_gincr:N \g__tag_para_main_begin_int
327   }
328   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
329   {
330     \int_gincr:N \g__tag_para_begin_int
331   }
332   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
333   {
334     \int_gincr:N \g__tag_para_main_end_int
335   }
336   \cs_set_protected:Npn \__tag_gincr_para_end_int:
337   {
338     \int_gincr:N \g__tag_para_end_int
339   }
340 }
341 \cs_new_protected:Npn \__tag_stop_para_ints:
342 {
343   \cs_set_eq:NN \__tag_gincr_para_main_begin_int: \prg_do_nothing:
344   \cs_set_eq:NN \__tag_gincr_para_begin_int: \prg_do_nothing:
345   \cs_set_eq:NN \__tag_gincr_para_main_end_int: \prg_do_nothing:
346   \cs_set_eq:NN \__tag_gincr_para_end_int: \prg_do_nothing:
347 }

```

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

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

```

348 \AddToHook{package/latex-lab-testphase-sec/after}
349 {
350   \cs_set_protected:Npn \@kernel@tag@hangfrom #1
351   {
352     \tagstructbegin{tag=\l__tag_para_tag_tl}
353     \__tag_gincr_para_begin_int:
354     \tagstructbegin{tag=Lbl}
355     \setbox\@tempboxa
356       \hbox
357       {
358         \bool_lazy_and:nnT
359         {\tag_if_active_p:}
360         {\g__tag_mode_lua_bool}
361         {\tagmcbegin{tag=Lbl}}
362         {#1}
363       }
364     \tag_stop:n{hangfrom}
365     \hangindent \wd\@tempboxa\noindent
366     \tag_start:n{hangfrom}
367     \tagmcbegin{}\box\@tempboxa\tagmcend\tagstructend\tagmcbegin{}
368   }
369 }

```

and two adaptations from the block module:

```

370 \AddToHook{package/latex-lab-testphase-block/after}
371 {
372   \cs_set_protected:Npn \__block_start_para_structure:n #1 {
373     \__block_debug_typeout:n
374     { @endpe = \legacy_if:nTF { @endpe }{true}{false}
375       \on@line }
376     \legacy_if:nF { @endpe }
377     {
378       \bool_if:NF \l__tag_para_flattened_bool
379       {
380         \__tag_gincr_para_main_begin_int:
381         \tag_struct_begin:n{tag=\l__tag_para_main_tag_tl}
382       }
383     }
384     \__tag_gincr_para_begin_int:
385     \__block_debug_typeout:n{increment~ P \on@line }
386     \tag_struct_begin:n
387     {
388       tag=\l__tag_para_tag_tl
389       ,attribute-class=\l__tag_para_attr_class_tl
390     }
391     \__tag_check_para_begin_show:nn {green}{#1}
392     \tag_mc_begin:n {}
393   }
394   \RemoveFromHook{para/end}[latex-lab-testphase-block]
395   \AddToHook{para/end}[latex-lab-testphase-block]
396   {

```

```

397 \bool_if:NT \l__tag_para_bool
398 {
399   \__tag_gincr_para_end_int:
400   \__block_debug_typeout:n{increment~ /P \on@line }
401   \tag_mc_end:
402   \__tag_check_para_end_show:nn {red}{}
403   \tag_struct_end:
404   \bool_if:NF \l__tag_para_flattened_bool
405   {
406     \__tag_gincr_para_main_end_int:
407     \tag_struct_end:
408   }
409 }
410 }
411 }
412

```

paratagging_␣(setup-key) These keys enable/disable locally paratagging, and the debug modus. It can affect the
paratagging-show_␣(setup-key) typesetting if **paratagging-show** is used. The small numbers are boxes and they have a
paratag_␣(setup-key) (small) height. The **paratag** key sets the tag used by the next automatic paratagging,
paratag_␣(tool-key) it can also be changed with **\tag_tool:n**
unittag_␣(tool-key)
para-flattened_␣(tool-key)

```

413 \keys_define:nn { __tag / setup }
414 {
415   paratagging .bool_set:N = \l__tag_para_bool,
416   paratagging-show .bool_set:N = \l__tag_para_show_bool,
417   paratag .tl_set:N = \l__tag_para_tag_tl
418 }
419 \keys_define:nn { tag / tool}
420 {
421   paratag .tl_set:N = \l__tag_para_tag_tl,
422   unittag .tl_set:N = \l__tag_para_main_tag_tl,
423   para-flattened .bool_set:N = \l__tag_para_flattened_bool
424 }

```

(End of definition for paratagging (setup-key) and others. These functions are documented on page 36.)

This fills the para hooks with the needed code.

```

425 \cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
426 % #1 color, #2 prefix
427 {
428   \bool_if:NT \l__tag_para_show_bool
429   {
430     \tag_mc_begin:n{artifact}
431     \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
432     \tag_mc_end:
433   }
434 }
435
436 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
437 % #1 color, #2 prefix
438 {
439   \bool_if:NT \l__tag_para_show_bool
440   {
441     \tag_mc_begin:n{artifact}

```



```

442         \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
443         \tag_mc_end:
444     }
445 }
446
447 \AddToHook{para/begin}
448 {
449     \bool_if:NT \l__tag_para_bool
450     {
451         \bool_if:NF \l__tag_para_flattened_bool
452         {
453             \__tag_gincr_para_main_begin_int:
454             \tag_struct_begin:n
455             {
456                 tag=\l__tag_para_main_tag_tl,
457             }
458         }
459         \__tag_gincr_para_begin_int:
460         \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
461         \__tag_check_para_begin_show:nn {green}{}
462         \tag_mc_begin:n {}
463     }
464 }
465 \AddToHook{para/end}
466 {
467     \bool_if:NT \l__tag_para_bool
468     {
469         \__tag_gincr_para_end_int:
470         \tag_mc_end:
471         \__tag_check_para_end_show:nn {red}{}
472         \tag_struct_end:
473         \bool_if:NF \l__tag_para_flattened_bool
474         {
475             \__tag_gincr_para_main_end_int:
476             \tag_struct_end:
477         }
478     }
479 }

```

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.

```

480 \AddToHook{enddocument/info}
481 {
482     \tag_if_active:F
483     {
484         \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
485     }
486     \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
487     {
488         \msg_error:nneee
489         {tag}
490         {para-hook-count-wrong}
491         {\int_use:N\g__tag_para_main_begin_int}

```

```

492         {\int_use:N\g__tag_para_main_end_int}
493         {text-unit}
494     }
495     \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
496     {
497         \msg_error:nneee
498         {tag}
499         {para-hook-count-wrong}
500         {\int_use:N\g__tag_para_begin_int}
501         {\int_use:N\g__tag_para_end_int}
502         {text}
503     }
504 }

```

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

```

505 \@ifpackageloaded{footmisc}
506   {\PackageWarning{tagpdf}{tagpdf~has~been~loaded~too~late!}} %
507   {\RequirePackage{latex-lab-testphase-new-or-1}}
508
509 \AddToHook{begindocument/before}
510 {
511     \providecommand\@kernel@taggsupport@@makecol{}
512     \providecommand\@kernel@before@cclv{}
513     \bool_if:NF \g__tag_mode_lua_bool
514     {
515         \cs_if_exist:NT \@kernel@before@footins
516         {
517             \tl_put_right:Nn \@kernel@before@footins
518             { \__tag_add_missing_mcs_to_stream:Nn \footins {footnote} }
519             \tl_put_right:Nn \@kernel@before@cclv
520             {
521                 \__tag_check_typeout_v:n {===>~In~\token_to_str:N \@makecol\c_space_tl\the\c@
522                 \__tag_add_missing_mcs_to_stream:Nn \@cclv {main}
523             }
524             \tl_put_right:Nn \@kernel@taggsupport@@makecol
525             {
526                 \__tag_check_typeout_v:n {===>~In~\token_to_str:N \@makecol\c_space_tl\the\c@
527                 \__tag_add_missing_mcs_to_stream:Nn \@outputbox {main}
528             }
529             \tl_put_right:Nn \@mult@ptagging@hook
530             {
531                 \__tag_check_typeout_v:n {===>~In~\string\page@sofar}
532                 \process@cols\mult@firstbox
533                 {
534                     \__tag_add_missing_mcs_to_stream:Nn \count@ {multicol}
535                 }
536                 \__tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol}
537             }
538         }
539     }
540 }
541 \end{package}

```

`\tagpdfparaOn` This two command switch para mode on and off. `\tagpdfsetup` could be used too but
`\tagpdfparaOff`

is longer. An alternative is `\tag_tool:n{para=false}`

```

542 \base\newcommand\tagpdfparaOn {}
543 \base\newcommand\tagpdfparaOff{}
544 \*package
545 \renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
546 \renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}
547 \keys_define:nn { tag / tool}
548 {
549     para .bool_set:N = \l__tag_para_bool,
550     para-flattened .bool_set:N = \l__tag_para_flattened_bool,
551 }

```

(End of definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 36.)

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

```

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

552 \NewDocumentCommand\tagpdfsuppressmarks{m}
553 {{\use:c{__tag_mc_disable_marks:} #1}}

```

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

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

```

554 \cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
555 \cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
556 \cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
557 \cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
558
559 \AddToHook{begindocument}
560 {
561     \cs_if_exist:NT \@kernel@before@head
562     {
563         \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
564         \tl_put_left:Nn \@kernel@after@head {\__tag_hook_kernel_after_head:}
565         \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
566         \tl_put_left:Nn \@kernel@after@foot {\__tag_hook_kernel_after_foot:}
567     }
568 }
569

```

```

570 \bool_new:N \g__tag_saved_in_mc_bool
571 \cs_new_protected:Npn \__tag_exclude_headfoot_begin:
572 {
573   \bool_set_false:N \l__tag_para_bool
574   \bool_if:NTF \g__tag_mode_lua_bool
575   {
576     \tag_mc_end_push:
577   }
578   {
579     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
580     \bool_gset_false:N \g__tag_in_mc_bool
581   }
582   \tag_mc_begin:n {artifact}
583   \tag_stop:n{headfoot}
584 }
585 \cs_new_protected:Npn \__tag_exclude_headfoot_end:
586 {
587   \tag_start:n{headfoot}
588   \tag_mc_end:
589   \bool_if:NTF \g__tag_mode_lua_bool
590   {
591     \tag_mc_begin_pop:n{ }
592   }
593   {
594     \bool_gset_eq:NN \g__tag_in_mc_bool \g__tag_saved_in_mc_bool
595   }
596 }

```

This version allows to use an Artifact structure

```

597 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
598 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
599 {
600   \bool_set_false:N \l__tag_para_bool
601   \bool_if:NTF \g__tag_mode_lua_bool
602   {
603     \tag_mc_end_push:
604   }
605   {
606     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
607     \bool_gset_false:N \g__tag_in_mc_bool
608   }
609   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
610   \tag_mc_begin:n {artifact=#1}
611   \tag_stop:n{headfoot}
612 }
613
614 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
615 {
616   \tag_start:n{headfoot}
617   \tag_mc_end:
618   \tag_struct_end:
619   \bool_if:NTF \g__tag_mode_lua_bool
620   {
621     \tag_mc_begin_pop:n{ }
622   }

```

```

623     {
624         \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
625     }
626 }

```

And now the keys

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

```

627 \keys_define:nn { __tag / setup }
628 {
629     exclude-header-footer .choice:,
630     exclude-header-footer / true .code:n =
631     {
632         \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
633         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
634         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
635         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
636     },
637     exclude-header-footer / pagination .code:n =
638     {
639         \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
640         \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
641         \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
642         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
643     },
644     exclude-header-footer / false .code:n =
645     {
646         \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
647         \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
648         \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
649         \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
650     },
651     exclude-header-footer .default:n = true,
652     exclude-header-footer .initial:n = true
653 }

```

(End of definition for `exclude-header-footer` (setup-key). This function is documented on page 37.)

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.

```

654 \hook_gput_code:nnn
655 {pdfannot/link/URI/before}
656 {tagpdf}
657 {
658     \tag_mc_end_push:
659     \tag_struct_begin:n { tag=Link }
660     \tag_mc_begin:n { tag=Link }
661     \pdfannot_dict_put:nne
662     { link/URI }
663     { StructParent }
664     { \tag_struct_parent_int: }
665 }

```

```

666
667 \hook_gput_code:nnn
668 {pdfannot/link/URI/after}
669 {tagpdf}
670 {
671   \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
672   \tag_mc_end:
673   \tag_struct_end:
674   \tag_mc_begin_pop:n{ }
675 }
676
677 \hook_gput_code:nnn
678 {pdfannot/link/GoTo/before}
679 {tagpdf}
680 {
681   \tag_mc_end_push:
682   \tag_struct_begin:n{tag=Link}
683   \tag_mc_begin:n{tag=Link}
684   \pdfannot_dict_put:nne
685     { link/GoTo }
686     { StructParent }
687     { \tag_struct_parent_int: }
688 }
689
690 \hook_gput_code:nnn
691 {pdfannot/link/GoTo/after}
692 {tagpdf}
693 {
694   \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
695   \tag_mc_end:
696   \tag_struct_end:
697   \tag_mc_begin_pop:n{ }
698 }
699 }
700
701 % "alternative descriptions " for PAX3. How to get better text here??
702 \pdfannot_dict_put:nnn
703 { link/URI }
704 { Contents }
705 { (url) }
706
707 \pdfannot_dict_put:nnn
708 { link/GoTo }
709 { Contents }
710 { (ref) }
711
</package>

```

Part III

The tagpdf-tree module

Commands trees and main dictionaries

Part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2023-12-18} {0.98r}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 </header>
```

1 Trees, pdfmanagement and finalization code

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

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

1.1 Check structure

__tag_tree_final_checks:

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

(End of definition for __tag_tree_final_checks:.)

1.2 Catalog: MarkInfo and StructTreeRoot

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

```

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

```

1.3 Writing the IDtree

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

```

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

```

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

```

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

```



```

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

```

1.4 Writing structure elements

The following commands are needed to write out the structure.

```

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

```

```

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

```

(End of definition for __tag_tree_write_structtreeroot:.)

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

```

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

```

(End of definition for __tag_tree_write_structelements:.)

1.5 ParentTree

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

```

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

```

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

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

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

```

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

```

```

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

```

(End of definition for \c@g__tag_parenttree_obj_int.)

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

\g__tag_parenttree_objr_tl

```

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

```

_tag_parenttree_add_objr:nn

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

```

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

```

(End of definition for _tag_parenttree_add_objr:nn.)

\l__tag_parenttree_content_tl

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

```

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

```

_tag_tree_fill_parenttree:

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

```

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

```

```

175 \int_step_inline:nnnn
176 {0}
177 {1}
178 { \prop_count:N \l__tag_tmpa_prop -1 }
179 {
180   \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
181   {% page#1:mcid##1:\l__tag_tmpa_tl :content
182     \tl_put_right:Ne \l__tag_parenttree_content_tl
183     {
184       \pdf_object_if_exist:eTF { __tag/struct/\l__tag_tmpa_tl }
185       {
186         \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
187       }
188       {
189         null
190       }
191       \c_space_tl
192     }
193   }
194   {
195     \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
196     {
197       \msg_warning:nn { tag } {tree-mcid-index-wrong}
198     }
199   }
200 }
201 \tl_put_right:Nn
202   \l__tag_parenttree_content_tl
203   {%[
204     ]^^J
205   }
206 }
207 }

```

(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.

```

208 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
209 {
210   \tl_set:Nn \l__tag_parenttree_content_tl
211   {
212     \lua_now:e
213     {
214       ltx.__tag.func.output_parenttree
215       (
216         \int_use:N\g_shipout_readonly_int
217       )
218     }
219   }
220 }

```

(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?

```

221 \cs_new_protected:Npn \__tag_tree_write_parenttree:
222 {
223   \bool_if:NTF \g__tag_mode_lua_bool
224   {
225     \__tag_tree_lua_fill_parenttree:
226   }
227   {
228     \__tag_tree_fill_parenttree:
229   }
230   \__tag_tree_parenttree_rerun_msg:
231   \tl_put_right:NV \l__tag_parenttree_content_tl \g__tag_parenttree_objr_tl
232   \pdf_object_write:nne { \__tag/tree/parenttree }{dict}
233   {
234     /Nums\c_space_tl [\l__tag_parenttree_content_tl]
235   }
236 }

```

(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.
237 \pdf_version_compare:NnT < {2.0}
238 {
239   \pdf_object_new:n { \__tag/tree/rolemap }
240 }

```

(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.

```

241 \pdf_version_compare:NnTF < {2.0}
242 {
243   \cs_new_protected:Npn \__tag_tree_write_rolemap:
244   {
245     \prop_map_inline:Nn \g__tag_role_rolemap_prop
246     {
247       \tl_if_eq:nnF {##1}{##2}
248       {
249         \pdfdict_gput:nne {g__tag_role/RoleMap_dict}
250         {##1}
251         {\pdf_name_from_unicode_e:n{##2}}
252       }
253     }
254     \pdf_object_write:nne { \__tag/tree/rolemap }{dict}
255     {
256       \pdfdict_use:n{g__tag_role/RoleMap_dict}
257     }
258   }
259 }

```

```

260     {
261         \cs_new_protected:Npn \__tag_tree_write_rolemap: {}
262     }
263

```

(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:

```

264 \cs_new_protected:Npn \__tag_tree_write_classmap:
265     {
266         \tl_clear:N \l__tag_tmpa_tl
267         \seq_gremove_duplicates:N \g__tag_attr_class_used_seq
268         \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
269         {
270             ##1\c_space_tl
271             <<
272             \prop_item:Nn
273             \g__tag_attr_entries_prop
274             {##1}
275             >>
276         }
277         \tl_set:Nc \l__tag_tmpa_tl
278         {
279             \seq_use:Nn
280             \l__tag_tmpa_seq
281             { \iow_newline: }
282         }
283         \tl_if_empty:NF
284         \l__tag_tmpa_tl
285         {
286             \pdf_object_new:n { __tag/tree/classmap }
287             \pdf_object_write:nne
288             { __tag/tree/classmap }
289             {dict}
290             { \l__tag_tmpa_tl }
291             \__tag_prop_gput:cne
292             { g__tag_struct_0_prop }
293             { ClassMap }
294             { \pdf_object_ref:n { __tag/tree/classmap } }
295         }
296     }

```

(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

```
297 \pdf_object_new:n { __tag/tree/namespaces }
```

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

__tag_tree_write_namespaces:

```
298 \cs_new_protected:Npn \__tag_tree_write_namespaces:
```

```
299 {
```

```
300   \pdf_version_compare:NnF < {2.0}
```

```
301   {
```

```
302     \prop_map_inline:Nn \g__tag_role_NS_prop
```

```
303     {
```

```
304       \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
```

```
305       {
```

```
306         \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
```

```
307         {
```

```
308           \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
```

```
309         }
```

```
310         \pdfdict_gput:nne{g__tag_role/Namespace_##1_dict}
```

```
311         {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
```

```
312       }
```

```
313       \pdf_object_write:nne{tag/NS/##1}{dict}
```

```
314       {
```

```
315         \pdfdict_use:n {g__tag_role/Namespace_##1_dict}
```

```
316       }
```

```
317     }
```

```
318     \pdf_object_write:nne {__tag/tree/namespaces}{array}
```

```
319     {
```

```
320       \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
```

```
321     }
```

```
322   }
```

```
323 }
```

(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:

```
324 \hook_new:n {tagpdf/finish/before}
```

```
325 \cs_new_protected:Npn \__tag_finish_structure:
```

```
326 {
```

```
327   \bool_if:NT\g__tag_active_tree_bool
```

```
328   {
```

```
329     \hook_use:n {tagpdf/finish/before}
```

```
330     \__tag_tree_final_checks:
```

```
331     \__tag_tree_write_parenttree:
```

```
332     \__tag_tree_write_idtree:
```

```
333     \__tag_tree_write_rolemap:
```

```
334     \__tag_tree_write_classmap:
```

```
335     \__tag_tree_write_namespaces:
```

```
336     \__tag_tree_write_structelements: %this is rather slow!!
```

```

337     \_tag_tree_write_structtreeroot:
338   }
339 }

```

(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.

```

340 \hook_gput_code:nnn{begindocument}{tagpdf}
341 {
342   \bool_if:NT\g__tag_active_tree_bool
343   {
344     \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
345     {
346       \pdfmanagement_add:nne
347       { Page }
348       { StructParents }
349       { \int_eval:n { \g_shipout_readonly_int } }
350     }
351   }
352 }
353 \</package>

```


Part IV

The **tagpdf-mc-shared** module

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

Part of the tagpdf package

1 Public Commands

<code>\tag_mc_begin:n</code>	<code>\tag_mc_begin:n{<key-values>}</code>
<code>\tag_mc_end:</code>	<code>\tag_mc_end:</code>

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

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

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

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

New: 2019-11-20

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

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

New: 2021-04-22

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

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

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

New: 2023-06-11

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

2 Public keys

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

<code>tag_␣(mc-key)</code>

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

<code>artifact_␣(mc-key)</code>

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

<code>raw_␣(mc-key)</code>

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

<code>alt_␣(mc-key)</code>

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

<code>actualtext_␣(mc-key)</code>

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

<code>label_␣(mc-key)</code>

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

<code>stash_␣(mc-key)</code>

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

The code is splitted into three parts: code shared by all engines, code specific to luamode and code not used by luamode.

3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2023-12-18} {0.98r}
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: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
```

```
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 65.)

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

`\tag_mc_artifact_group_end:`

```

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 65.)

`\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 66.)

`\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:Nc \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 65.)

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:nn { __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 66.)

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} {2023-12-18} {0.98r}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2023-12-18} {0.98r}
8 {part of tagpdf - debugging code related to marking chunks - generic mode}
9 </debug>
```

1.1 Variables

```
10 <*generic>
```

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

```
11 \tl_new:N \l__tag_mc_ref_abspage_tl
```

(End of definition for `\l__tag_mc_ref_abspage_tl`.)

`\l__tag_mc_tmpa_tl` temporary variable

```
12 \tl_new:N \l__tag_mc_tmpa_tl
```

(End of definition for `\l__tag_mc_tmpa_tl`.)

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

```
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:Nc
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         %store
130         \__tag_mc_store:eee
131         {
132           \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
133         }
134         { \int_eval:n{\c@g__tag_MCID_abs_int} }
135         {

```

```

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 und is currently either `main` for the main galley, `footnote` for footnote note text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```

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 therefor guarded by the mc-boolean.

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

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

```

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` 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 65.)

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

```

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 number value for the MCID: they must be numbered by page starting with 0 and then successively. The first argument is the tag, e.g. P or Span, the second is used to pass more properties. 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.
 __tag_mc_bdc_mcid:n
 __tag_mc_handle_mcid:nn
 __tag_mc_handle_mcid:VV

```

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:Nx \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

```

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 65.)

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)
alt_(mc-key)
actualtext_(mc-key)
label_(mc-key)
artifact_(mc-key)
453 <*generic>
454 \keys_define:nn { _tag / mc }
455 {
456   tag .code:n = % the name (H,P,Span) etc
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 66.)

Part VI

The tagpdf-mc-luacode module

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

Part of the tagpdf package

The code is splitted into three parts: code shared by all engines, code specific to luamode and code not used by luamode.

1 Marked content code – luamode code

luamode uses attributes to mark mc-chunks. The two attributes used are defined in the backend file. The backend also load the lua file, as it can contain functions needed elsewhere. The attributes for mc are global (between 0.6 and 0.81 they were local but this was reverted). The attributes are setup only in lua, and one should use the lua functions to set and get them.

`g_@@_mc_type_attr`: the value represent the type

`g_@@_mc_cnt_attr`: will hold the `\c@g_@@_MCID_abs_int` value

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

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

`tag` : the type (a string)

`raw` : more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2023-12-18} {0.98r}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2023-12-18} {0.98r}
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.

```

\_tag_mc_if_in:TF 66 \prg_new_conditional:Nnn \_tag_mc_if_in: {p,T,F,TF}
\_tag_mc_if_in_p: 67 {
\_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 65.)

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

```

\_tag_mc_lua_set_mc_type_attr:o 81 \cs_new:Nn \_tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
\_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:Nl\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
 __tag_mc_insert_mcid_single_kids:n kids we need a variant for the case that it is the only kid, to get the array right

```

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
 __tag_mc_handle_stash:e 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 65.)

\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 65.)

`\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 66.)

`__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:Noo
                    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 66.)

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.

<code>\tag_struct_gput:nnn</code>	<code>\tag_struct_gput:nnn{<structure number>}{<keyword>}{<value>}</code>
-----------------------------------	---

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

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

<code>stash_<struct-key></code>	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.
---------------------------------------	---

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

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

<code>title_<struct-key></code> <code>title-o_<struct-key></code>	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.
--	---

<code>alt_<struct-key></code>	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.
-------------------------------------	---

<code>actualtext_□(struct-key)</code>	This key inserts an <code>/ActualText</code> 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.
---	--

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

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

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

<code>AF_□(struct-key)</code>	<code>AF = <object name></code>
<code>AFref_□(struct-key)</code>	<code>AFref = <object reference></code>
<code>AFinline_□(struct-key)</code>	<code>AF-inline = <text content></code>
<code>AFinline-o_□(struct-key)</code>	These keys allows to reference an associated file in the structure element. The value <code><object name></code> should be the name of an object pointing to the <code>/Filespec</code> dictionary as expected by <code>\pdf_object_ref:n</code> from a current <code>l3kernel</code> .
<code>texsource</code>	
<code>mathml</code>	

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

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

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

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2023-12-18} {0.98r}
4 {part of tagpdf - code related to storing structure}
5 </header>
```

3 Variables

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

```
6 <base>\newcounter { g__tag_struct_abs_int }
7 <base>\int_gzero:N \c@g__tag_struct_abs_int
```

(End of definition for `\c@g__tag_struct_abs_int`.)

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

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

(End of definition for `\g__tag_struct_objR_seq`.)

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

```
10 \__tag_prop_new:N \g__tag_struct_cont_mc_prop
```

(End of definition for `\g__tag_struct_cont_mc_prop`.)

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

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

(End of definition for `\g__tag_struct_stack_seq`.)

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

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

(End of definition for `\g__tag_struct_tag_stack_seq`.)

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

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

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

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

The data of the StructTreeRoot and the StructElem are in properties: `\g_@@_struct_0_prop` for the root and `\g_@@_struct_N_prop`, $N \geq 1$ for the other.

This creates quite a number of properties, so perhaps we will have to do this more efficiently in the future.

All properties have at least the keys

Type StructTreeRoot or StructElem

and the keys from the two following lists (the root has a special set of properties). the values of the prop should be already escaped properly when the entries are created (title, lange, alt, E, actualtext)

\c__tag_struct_StructTreeRoot_entries_seq
 \c__tag_struct_StructElem_entries_seq

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

```

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

```

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

3.1 Variables used by the keys

\g__tag_struct_tag_tl
 \g__tag_struct_tag_NS_tl
 \l__tag_struct_roletag_tl
 \g__tag_struct_roletag_NS_tl

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

```

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

```

(End of definition for \g__tag_struct_tag_tl and others.)

`\l__tag_struct_key_label_tl` This will hold the label value.

```

61 \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

```

62 \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.

```

63 </package>
64 <base>\prop_new:N \g__tag_struct_dest_num_prop
65 <*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.

```

66 \prop_new:N \g__tag_struct_ref_by_dest_prop

(End of definition for \g__tag_struct_ref_by_dest_prop.)
```

4 Commands

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

```

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

```

82     }
83   }
84 </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:

```

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

```

95 <*package>
96 \tl_gset:Nn \g__tag_struct_stack_current_tl {0}

```

`__tag_pdf_name_e:n`

```

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

```

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

`g__tag_struct_0_prop`
`g__tag_struct_kids_0_seq`

```

99 <*package>
100 __tag_prop_new:c { g__tag_struct_0_prop }
101 __tag_new_output_prop_handler:n {0}
102 __tag_seq_new:c { g__tag_struct_kids_0_seq }
103
104 __tag_struct_prop_gput:nne
105 { 0 }
106 { Type }
107 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
108
109 __tag_struct_prop_gput:nne
110 { 0 }
111 { S }
112 { \pdf_name_from_unicode_e:n {StructTreeRoot} }
113
114 __tag_struct_prop_gput:nne

```

```

115 { 0 }
116 { rolemap }
117 { {StructTreeRoot}{pdf} }
118
119 \__tag_struct_prop_gput:nne
120 { 0 }
121 { parentrole }
122 { {StructTreeRoot}{pdf} }
123

```

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

```

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

```

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

```

132 <debug>\prop_new:c { g__tag_struct_debug_0_prop }
133 <debug>\seq_new:c { g__tag_struct_debug_kids_0_seq }
134 <debug>\prop_gset_eq:cc { g__tag_struct_debug_0_prop }{ g__tag_struct_0_prop }
135 <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
136 <*package>
137 \cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
138 {
139   (
140     ID.
141     \prg_replicate:nn
142     { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
143     { 0 }
144     \int_to_arabic:n { #1 }
145   )
146 }

```

(End of definition for __tag_struct_get_id:n.)

4.3 Filling in the tag info

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

```

147 \pdf_version_compare:NnTF < {2.0}
148 {

```

```

149 \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
150   {%#1 structure number, #2 tag, #3 NS
151   {
152     \__tag_struct_prop_gput:nne
153     { #1 }
154     { S }
155     { \pdf_name_from_unicode_e:n {#2} } %
156   }
157 }
158 {
159 \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
160   {
161     \__tag_struct_prop_gput:nne
162     { #1 }
163     { S }
164     { \pdf_name_from_unicode_e:n {#2} } %
165     \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
166     {
167       \__tag_struct_prop_gput:nne
168       { #1 }
169       { NS }
170       { \l__tag_get_tmpc_tl } %
171     }
172   }
173 }
174 \cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eVV}
(End of definition for \__tag_struct_set_tag_info:nnn.)

```

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

```

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

```

4.4 Handlings kids

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

(through an OBJR object).

_tag_struct_kid_mc_gput_right:nn
_tag_struct_kid_mc_gput_right: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!

```

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

```

224 <package>\cs_new_protected:Npn\_tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent s
225 <debug>\cs_set_protected:Npn\_tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent str
226 {
227   \_tag_seq_gput_right:ce
228   { g__tag_struct_kids_#1_seq }
229   {
230     \pdf_object_ref:n { \_tag/struct/#2 }

```

```

231     }
232   <debug>      \seq_gput_right:cn
233   <debug>      { g__tag_struct_debug_kids_#1_seq }
234   <debug>      {
235     <debug>      Struct~#2
236   <debug>      }
237   }
238
239   <package>\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {ee}

```

(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

```

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

```

264   \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
265   {
266     \seq_gpop_left:NN #1 \l__tag_tmpa_tl
267     \regex_replace_once:nnN
268     { \c{\__tag_mc_insert_mcid_kids:n} }
269     { \c{\__tag_mc_insert_mcid_single_kids:n} }
270     \l__tag_tmpa_tl
271     \seq_gput_left:NV #1 \l__tag_tmpa_tl

```

```

272 }
273
274 \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.

```

275 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
276 {
277   \bool_if:NF\g__tag_mode_lua_bool
278   {
279     \seq_clear:N \l__tag_tmpa_seq
280     \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
281     { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
282     %\seq_show:c { g__tag_struct_kids_#1_seq }
283     %\seq_show:N \l__tag_tmpa_seq
284     \seq_remove_all:Nn \l__tag_tmpa_seq {}
285     %\seq_show:N \l__tag_tmpa_seq
286     \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
287   }
288
289   \int_case:nnF
290   {
291     \seq_count:c
292     {
293       g__tag_struct_kids_#1_seq
294     }
295   }
296   {
297     { 0 }
298     { } %no kids, do nothing
299     { 1 } % 1 kid, insert
300     {
301       % in this case we need a special command in
302       % luamode to get the array right. See issue #13
303       \bool_if:NT\g__tag_mode_lua_bool
304       {
305         \__tag_struct_exchange_kid_command:c
306         {g__tag_struct_kids_#1_seq}
307       }
308       \__tag_struct_prop_gput:nne
309       {#1}
310       {K}
311       {
312         \seq_item:cn
313         {
314           g__tag_struct_kids_#1_seq
315         }
316         {1}
317       }
318     } %
319   }
320   { %many kids, use an array

```

```

321     \_tag_struct_prop_gput:nne
322     {#1}
323     {K}
324     {
325     [
326     \seq_use:cn
327     {
328     g__tag_struct_kids_#1_seq
329     }
330     {
331     \c_space_tl
332     }
333     ]
334     }
335 }
336 }
337

```

(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.

```

338 \cs_new_protected:Npn \_tag_struct_get_dict_content:nN #1 #2 %#1: structure num
339 {
340     \tl_clear:N #2
341     \seq_map_inline:cn
342     {
343     c__tag_struct_
344     \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
345     _entries_seq
346     }
347     {
348     \tl_put_right:Ne
349     #2
350     {
351     \prop_if_in:cnT
352     { g__tag_struct_#1_prop }
353     { ##1 }
354     {
355     \c_space_tl/##1~

```

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

```

356     \cs_if_exist_use:cTF {__tag_struct_format_##1:e}
357     {
358     { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } }
359     }
360     {
361     \prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
362     }
363     }
364 }

```

```

365     }
366 }

```

(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:

```

367 \cs_new:Nn __tag_struct_format_Ref:n{[#1]}
368 \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.

```

369 \cs_new_protected:Npn __tag_struct_write_obj:n #1 % #1 is the struct num
370 {
371   \pdf_object_if_exist:nTF { __tag/struct/#1 }
372   {

```

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.

```

373     \prop_get:cnNF { g__tag_struct_#1_prop } {P}\l__tag_tmpb_tl
374     {
375       \prop_gput:cne { g__tag_struct_#1_prop } {P}{\pdf_object_ref:n { __tag/struct/0 }
376       \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
377       \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
378       {
379         \msg_warning:nnee
380           {tag}
381           {struct-orphan}
382           { #1 }
383           {\seq_count:c{g__tag_struct_kids_#1_seq}}
384       }
385     }
386     __tag_struct_fill_kid_key:n { #1 }
387     __tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
388     \exp_args:Ne
389       \pdf_object_write:nne
390       { __tag/struct/#1 }
391       {dict}
392       {
393         \l__tag_tmpa_tl\c_space_tl
394         /ID~__tag_struct_get_id:n{#1}
395       }
396
397   }
398   {
399     \msg_error:nnn { tag } { struct-no-objnum } { #1}
400   }
401 }

```

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

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

Annotations used as structure content must

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

For a link this looks like this

```

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

402 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat
403                                     %#2 structparent number
404 {
405   \bool_if:NT \g__tag_active_struct_bool
406   {
407     %get the number of the parent structure:
408     \seq_get:NNF
409     \g__tag_struct_stack_seq
410     \l__tag_struct_stack_parent_tmpa_tl
411     {
412       \msg_error:nn { tag } { struct-faulty-nesting }
413     }
414     %put the obj number of the annot in the kid entry, this also creates
415     %the OBJR object
416     \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
417     \__tag_struct_kid_OBJR_gput_right:eee
418     {
419       \l__tag_struct_stack_parent_tmpa_tl
420     }
421     {
422       #1 %
423     }
424     {
425       \pdf_pageobject_ref:n { \__tag_property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }
426     }
427     % add the parent obj number to the parent tree:
428     \exp_args:Nne
429     \__tag_parenttree_add_objr:nn
430     {
431       #2
432     }
433     {
434       \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
435     }
436     % increase the int:
437     \stepcounter{ g__tag_parenttree_obj_int }

```

```

438     }
439 }

```

(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`.

```

440 \cs_new:Npn \_tag_get_data_struct_tag:
441 {
442   \exp_args:Ne
443   \tl_tail:n
444   {
445     \prop_item:cn {g__tag_struct_g__tag_struct_stack_current_tl _prop}{S}
446   }
447 }

```

(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`.

```

448 \cs_new:Npn \_tag_get_data_struct_id:
449 {
450   \_tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
451 }
452 </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

```

453 <*base>
454 \cs_new:Npn \_tag_get_data_struct_num:
455 {
456   \g__tag_struct_stack_current_tl
457 }
458 </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.

```

459 <*base>
460 \cs_new:Npn \_tag_get_data_struct_counter:
461 {
462   \int_use:N \c@g__tag_struct_abs_int
463 }
464 </base>

```

(End of definition for _tag_get_data_struct_counter:.)

5 Keys

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

```

label_(struct-key)
stash_(struct-key) 465 (*package)
parent_(struct-key) 466 \keys_define:nn { __tag / struct }
tag_(struct-key)    467 {
title_(struct-key)  468   label .tl_set:N      = \l__tag_struct_key_label_tl,
title-o_(struct-key) 469   stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
alt_(struct-key)    470   parent .code:n      =
actualtext_(struct-key) 471   {
lang_(struct-key)   472     \bool_lazy_and:nnTF
ref_(struct-key)    473     {
E_(struct-key)      474       \prop_if_exist_p:c { g__tag_struct\_int_eval:n {#1}_prop }
475     }
476     {
477       \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
478     }
479     { \tl_set:Nc \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
480     {
481       \msg_warning:nnee { tag } { struct-unknown }
482       { \int_eval:n {#1} }
483       { parent-key-ignored }
484     }
485   },
486   parent .default:n    = {-1},
487   tag .code:n          = % S property
488   {
489     \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Nc\g__tag_role_tags_NS_prop{
490     \tl_gset:Nc \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
491     \tl_gset:Nc \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
492     \__tag_check_structure_tag:N \g__tag_struct_tag_tl
493   },
494   title .code:n        = % T property
495   {
496     \str_set_convert:Nnnn
497     \l__tag_tmpa_str
498     { #1 }
499     { default }
500     { utf16/hex }
501     \__tag_struct_prop_gput:nne
502     { \int_use:N \c@g__tag_struct_abs_int }
503     { T }
504     { <\l__tag_tmpa_str> }
505   },
506   title-o .code:n      = % T property
507   {
508     \str_set_convert:Nonn
509     \l__tag_tmpa_str
510     { #1 }
511     { default }
512     { utf16/hex }

```



```

513     \_tag_struct_prop_gput:nne
514     { \int_use:N \c@g__tag_struct_abs_int }
515     { T }
516     { <\l__tag_tmpa_str> }
517 },
518 alt .code:n      = % Alt property
519 {
520   \tl_if_empty:oF{#1}
521   {
522     \str_set_convert:Noon
523     \l__tag_tmpa_str
524     { #1 }
525     { default }
526     { utf16/hex }
527     \_tag_struct_prop_gput:nne
528     { \int_use:N \c@g__tag_struct_abs_int }
529     { Alt }
530     { <\l__tag_tmpa_str> }
531   }
532 },
533 alttext .meta:n = {alt=#1},
534 actualtext .code:n = % ActualText property
535 {
536   \tl_if_empty:oF{#1}
537   {
538     \str_set_convert:Noon
539     \l__tag_tmpa_str
540     { #1 }
541     { default }
542     { utf16/hex }
543     \_tag_struct_prop_gput:nne
544     { \int_use:N \c@g__tag_struct_abs_int }
545     { ActualText }
546     { <\l__tag_tmpa_str>}
547   }
548 },
549 lang .code:n      = % Lang property
550 {
551   \_tag_struct_prop_gput:nne
552   { \int_use:N \c@g__tag_struct_abs_int }
553   { Lang }
554   { (#1) }
555 },

```

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

```

556 ref .code:n      = % ref property
557 {
558   \tl_clear:N\l__tag_tmpa_tl
559   \clist_map_inline:on {#1}
560   {
561     \tl_put_right:Ne \l__tag_tmpa_tl
562     {~\_tag_property_ref:en{tagpdfstruct-##1}{tagstructobj} }
563   }
564   \_tag_struct_gput_data_ref:ee
565   { \int_use:N \c@g__tag_struct_abs_int } {\l__tag_tmpa_tl}

```

```

566     },
567     E.code:n          = % E property
568     {
569         \str_set_convert:Nnon
570         \l__tag_tmpa_str
571         { #1 }
572         { default }
573         { utf16/hex }
574         \__tag_struct_prop_gput:nne
575         { \int_use:N \c@g__tag_struct_abs_int }
576         { E }
577         { <\l__tag_tmpa_str> }
578     },
579 }

```

(End of definition for label (struct-key) and others. These functions are documented on page 96.)

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

```

580 \int_new:N\g__tag_struct_AFobj_int

581 \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
\g__tag_struct_AFobj_int 582 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
583 % #1 content, #2 extension
584 {
585     \tl_if_empty:nF{#1}
586     {
587         \group_begin:
588         \int_gincr:N \g__tag_struct_AFobj_int
589         \pdffile_embed_stream:neN
590         {#1}
591         {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
592         \l__tag_tmpa_tl
593         \__tag_struct_add_AF:ee
594         { \int_use:N \c@g__tag_struct_abs_int }
595         { \l__tag_tmpa_tl }
596         \__tag_struct_prop_gput:nne
597         { \int_use:N \c@g__tag_struct_abs_int }
598         { AF }
599         {
600             [
601                 \tl_use:c
602                 { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
603             ]
604         }
605         \group_end:
606     }
607 }

```

```

608
609 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
610 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object reference
611 {
612     \tl_if_exist:cTF
613     {
614         g__tag_struct_#1_AF_tl
615     }
616     {
617         \tl_gput_right:ce
618         { g__tag_struct_#1_AF_tl }
619         { \c_space_tl #2 }
620     }
621     {
622         \tl_new:c
623         { g__tag_struct_#1_AF_tl }
624         \tl_gset:ce
625         { g__tag_struct_#1_AF_tl }
626         { #2 }
627     }
628 }
629 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
630 \keys_define:nn { __tag / struct }
631 {
632     AF .code:n          = % AF property
633     {
634         \pdf_object_if_exist:eTF {#1}
635         {
636             \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e
637             \__tag_struct_prop_gput:nne
638             { \int_use:N \c@g__tag_struct_abs_int }
639             { AF }
640             {
641                 [
642                     \tl_use:c
643                     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
644                 ]
645             }
646         }
647         {
648             % message?
649         }
650     },
651     AFref .code:n       = % AF property
652     {
653         \tl_if_empty:eF {#1}
654         {
655             \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
656             \__tag_struct_prop_gput:nne
657             { \int_use:N \c@g__tag_struct_abs_int }
658             { AF }
659             {
660                 [
661                     \tl_use:c

```

```

662         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_t1 }
663     ]
664 }
665 }
666 },
667 ,AFinline .code:n =
668 {
669     \__tag_struct_add_inline_AF:nn {#1}{txt}
670 }
671 ,AFinline-o .code:n =
672 {
673     \__tag_struct_add_inline_AF:on {#1}{txt}
674 }
675 ,texsource .code:n =
676 {
677     \group_begin:
678     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
679     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
680     \__tag_struct_add_inline_AF:on {#1}{tex}
681     \group_end:
682 }
683 ,mathml .code:n =
684 {
685     \group_begin:
686     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
687     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
688     \__tag_struct_add_inline_AF:on {#1}{xml}
689     \group_end:
690 }
691 }

```

(End of definition for AF (struct-key) and others. These functions are documented on page 97.)

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!

```

692 \keys_define:nn { __tag / setup }
693 {
694     root-AF .code:n =
695     {
696         \pdf_object_if_exist:nTF {#1}
697         {
698             \__tag_struct_add_AF:ee { 0 }{\pdf_object_ref:n {#1}}
699             \__tag_struct_prop_gput:nne
700             { 0 }
701             { AF }
702             {
703                 [
704                     \tl_use:c
705                     { g__tag_struct_0_AF_t1 }
706                 ]
707             }
708         }
709     }
710 }

```

```

711     }
712   },
713 }
714 </package>

```

(End of definition for root-AF (setup-key). This function is documented on page 98.)

6 User commands

`\tag_struct_begin:n`

`\tag_struct_end:`

```

715 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
716 <base>\cs_new_protected:Npn \tag_struct_end: {}
717 <base>\cs_new_protected:Npn \tag_struct_end:n {}
718 *package| debug
719 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
720 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
721 {
722 <package>\__tag_check_if_active_struct:T
723 <debug>\__tag_check_if_active_struct:TF
724 {
725   \group_begin:
726   \int_gincr:N \c@g__tag_struct_abs_int
727   \__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
728 <debug>   \prop_new:c { g__tag_struct_debug_\int_eval:n { \c@g__tag_struct_abs_int }_prop
729   \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
730   \__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
731 <debug>   \seq_new:c { g__tag_struct_debug_kids_\int_eval:n { \c@g__tag_struct_abs_int }_s
732   \exp_args:Ne
733   \pdf_object_new:n
734   { \__tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
735   \__tag_struct_prop_gput:nnn
736   { \int_use:N \c@g__tag_struct_abs_int }
737   { Type }
738   { /StructElem }
739   \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
740   \keys_set:nn { __tag / struct } { #1 }
741   \__tag_struct_set_tag_info:eVV
742   { \int_use:N \c@g__tag_struct_abs_int }
743   \g__tag_struct_tag_tl
744   \g__tag_struct_tag_NS_tl
745   \__tag_check_structure_has_tag:n { \int_use:N \c@g__tag_struct_abs_int }
746   \tl_if_empty:NF
747   \l__tag_struct_key_label_tl
748   {
749     \__tag_property_record:eV
750     {tagpdfstruct-\l__tag_struct_key_label_tl}
751     \c__tag_property_struct_clist
752   }

```

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

```

753   \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
754   {

```

```

755         \seq_get:NNF
756         \g__tag_struct_stack_seq
757         \l__tag_struct_stack_parent_tmpa_tl
758         {
759             \msg_error:nn { tag } { struct-faulty-nesting }
760         }
761     }
762     \seq_gpush:NV \g__tag_struct_stack_seq          \c@g__tag_struct_abs_int
763     \__tag_role_get:VVNN
764     \g__tag_struct_tag_tl
765     \g__tag_struct_tag_NS_tl
766     \l__tag_struct_roletag_tl
767     \l__tag_struct_roletag_NS_tl

```

to target role and role NS

```

768     \__tag_struct_prop_gput:nne
769     { \int_use:N \c@g__tag_struct_abs_int }
770     { rolemap }
771     {
772         {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
773     }

```

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!

```

774     \str_case:VnTF \l__tag_struct_roletag_tl
775     {
776         {Part} {}
777         {Div} {}
778         {NonStruct} {}
779     }
780     {
781         \prop_get:cnNT
782         { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
783         { parentrole }
784         \l__tag_get_tmpc_tl
785         {
786             \__tag_struct_prop_gput:nno
787             { \int_use:N \c@g__tag_struct_abs_int }
788             { parentrole }
789             {
790                 \l__tag_get_tmpc_tl
791             }
792         }
793     }
794     {
795         \__tag_struct_prop_gput:nne
796         { \int_use:N \c@g__tag_struct_abs_int }
797         { parentrole }
798         {
799             {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
800         }
801     }
802     \seq_gpush:Ne \g__tag_struct_tag_stack_seq

```

```

803     {{\g__tag_struct_tag_t1}{\l__tag_struct_roletag_t1}}
804     \tl_gset:NV \g__tag_struct_stack_current_t1 \c@g__tag_struct_abs_int
805     %\seq_show:N \g__tag_struct_stack_seq
806     \bool_if:NF
807         \l__tag_struct_elem_stash_bool
808     {

```

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!

```

809         \__tag_struct_get_parentrole:eNN
810         {\l__tag_struct_stack_parent_tmpa_t1}
811         \l__tag_get_parent_tmpa_t1
812         \l__tag_get_parent_tmpb_t1
813         \__tag_check_parent_child:VVVVN
814         \l__tag_get_parent_tmpa_t1
815         \l__tag_get_parent_tmpb_t1
816         \g__tag_struct_tag_t1
817         \g__tag_struct_tag_NS_t1
818         \l__tag_parent_child_check_t1
819         \int_compare:nNnT {\l__tag_parent_child_check_t1}<0
820         {
821             \prop_get:cnN
822             { \g__tag_struct_ \l__tag_struct_stack_parent_tmpa_t1 _prop}
823             {S}
824             \l__tag_tmpa_t1
825             \msg_warning:nneee
826             { tag }
827             {role-parent-child}
828             { \l__tag_get_parent_tmpa_t1/\l__tag_get_parent_tmpb_t1 }
829             { \g__tag_struct_tag_t1/\g__tag_struct_tag_NS_t1 }
830             { not~allowed~
831                 (struct~\l__tag_struct_stack_parent_tmpa_t1,~\l__tag_tmpa_t1
832                 \c_space_t1-->~struct~\int_eval:n {\c@g__tag_struct_abs_int})
833             }
834             \cs_set_eq:NN \l__tag_role_remap_tag_t1 \g__tag_struct_tag_t1
835             \cs_set_eq:NN \l__tag_role_remap_NS_t1 \g__tag_struct_tag_NS_t1
836             \__tag_role_remap:
837             \cs_gset_eq:NN \g__tag_struct_tag_t1 \l__tag_role_remap_tag_t1
838             \cs_gset_eq:NN \g__tag_struct_tag_NS_t1 \l__tag_role_remap_NS_t1
839             \__tag_struct_set_tag_info:eVV
840             { \int_use:N \c@g__tag_struct_abs_int }
841             \g__tag_struct_tag_t1
842             \g__tag_struct_tag_NS_t1
843         }

```

Set the Parent.

```

844         \__tag_struct_prop_gput:nne
845         { \int_use:N \c@g__tag_struct_abs_int }
846         { P }
847         {
848             \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_t1 }
849         }
850         %record this structure as kid:

```

```

851         %\tl_show:N \g__tag_struct_stack_current_tl
852         %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
853         \__tag_struct_kid_struct_gput_right:ee
854         { \l__tag_struct_stack_parent_tmpa_tl }
855         { \g__tag_struct_stack_current_tl }
856         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
857         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
858     }

```

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.

```

859 <debug>         \prop_gset_eq:cc
860 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
861 <debug>         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
862 <debug>         \prop_gput:cne
863 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
864 <debug>         { P }
865 <debug>         {
866 <debug>             \bool_if:NTF \l__tag_struct_elem_stash_bool
867 <debug>             {no~parent:~stashed}
868 <debug>             {
869 <debug>                 parent~structure:~\l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
870 <debug>                 \l__tag_get_parent_tmpa_tl
871 <debug>             }
872 <debug>         }
873 <debug>         \prop_gput:cne
874 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
875 <debug>         { NS }
876 <debug>         { \g__tag_struct_tag_NS_tl }

877         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
878         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
879 <debug> \__tag_debug_struct_begin_insert:n { #1 }
880     \group_end:
881 }
882 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
883 }
884 <package>\cs_set_protected:Nn \tag_struct_end:
885 <debug>\cs_set_protected:Nn \tag_struct_end:
886     { %take the current structure num from the stack:
887       %the objects are written later, lua mode hasn't all needed info yet
888       %\seq_show:N \g__tag_struct_stack_seq
889 <package>\__tag_check_if_active_struct:T
890 <debug>\__tag_check_if_active_struct:TF
891     {
892         \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
893         \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
894         {
895             \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
896         }
897         { \__tag_check_no_open_struct: }
898         % get the previous one, shouldn't be empty as the root should be there
899         \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
900         {

```



```

901         \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
902     }
903     {
904         \__tag_check_no_open_struct:
905     }
906     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
907     {
908         \tl_gset:Ne \g__tag_struct_tag_tl
909         { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
910         \prop_get:NVNT\g__tag_role_tags_NS_prop \g__tag_struct_tag_tl\l__tag_tmpa_tl
911         {
912             \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
913         }
914     }
915     <debug>\__tag_debug_struct_end_insert:
916     }
917     <debug>{\__tag_debug_struct_end_ignore:}
918     }
919
920     \cs_set_protected:Npn \tag_struct_end:n #1
921     {
922     <debug> \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
923     \tag_struct_end:
924     }
925     </package|debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 95.)

\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.

```

926     <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
927     <*package|debug>
928     \cs_set_protected:Npn \tag_struct_use:n #1 %1 is the label
929     {
930         \__tag_check_if_active_struct:T
931         {
932             \prop_if_exist:cTF
933             { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop }
934             {
935                 \__tag_check_struct_used:n {#1}
936                 %add the label structure as kid to the current structure (can be the root)
937                 \__tag_struct_kid_struct_gput_right:ee
938                 { \g__tag_struct_stack_current_tl }
939                 { \__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0} }
940                 %add the current structure to the labeled one as parents
941                 \__tag_prop_gput:cne
942                 { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
943                 { P }
944                 {
945                     \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
946                 }
947             }
948         }
949     }
950     debug code
951     <debug> \prop_gput:cne

```

```

948 <debug>          { g__tag_struct_debug\___tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}}{
949 <debug>          { P }
950 <debug>          {
951 <debug>              parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
952 <debug>              \g__tag_struct_tag_tl
953 <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:

```

954          \__tag_struct_get_parentrole:eNN
955          {\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}}{0}}
956          \l__tag_tmpa_tl
957          \l__tag_tmpb_tl
958          \__tag_check_parent_child:VVVVN
959          \g__tag_struct_tag_tl
960          \g__tag_struct_tag_NS_tl
961          \l__tag_tmpa_tl
962          \l__tag_tmpb_tl
963          \l__tag_parent_child_check_tl
964          \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
965          {
966              \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
967              \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
968              \__tag_role_remap:
969              \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
970              \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
971              \__tag_struct_set_tag_info:eVV
972              { \int_use:N \c@g__tag_struct_abs_int }
973              \g__tag_struct_tag_tl
974              \g__tag_struct_tag_NS_tl
975          }
976      }
977      {
978          \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
979      }
980  }
981 }
982 </package | debug>

```

(End of definition for \tag_struct_use:n. This function is documented on page 95.)

\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.

```

983 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
984 <*package | debug>
985 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
986 {
987     \__tag_check_if_active_struct:T
988     {
989         \prop_if_exist:cTF
990         { g__tag_struct_#1_prop } %
991         {
992             \prop_get:cnNT

```

```

993         {g__tag_struct_#1_prop}
994         {P}
995         \l__tag_tmpa_tl
996         {
997             \msg_warning:nnn { tag } {struct-used-twice} {#1}
998         }
999 %add the \#1 structure as kid to the current structure (can be the root)
1000 \__tag_struct_kid_struct_gput_right:ee
1001     { \g__tag_struct_stack_current_tl }
1002     { #1 }
1003 %add the current structure to \#1 as parent
1004 \__tag_struct_prop_gput:nne
1005     { #1 }
1006     { P }
1007     {
1008         \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
1009     }
1010 <debug>         \prop_gput:cne
1011 <debug>         { g__tag_struct_debug_#1_prop }
1012 <debug>         { P }
1013 <debug>         {
1014 <debug>             parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1015 <debug>             \g__tag_struct_tag_tl
1016 <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:

```

1017         \__tag_struct_get_parentrole:eNN
1018         {#1}
1019         \l__tag_tmpa_tl
1020         \l__tag_tmpp_tl
1021 \__tag_check_parent_child:VVVVN
1022     \g__tag_struct_tag_tl
1023     \g__tag_struct_tag_NS_tl
1024     \l__tag_tmpa_tl
1025     \l__tag_tmpp_tl
1026     \l__tag_parent_child_check_tl
1027 \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1028 {
1029     \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1030     \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1031     \__tag_role_remap:
1032     \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1033     \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1034     \__tag_struct_set_tag_info:eVV
1035     { \int_use:N \c@g__tag_struct_abs_int }
1036     \g__tag_struct_tag_tl
1037     \g__tag_struct_tag_NS_tl
1038 }
1039 }
1040 {
1041     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1042 }
1043 }

```

```

1044 }
1045 </package | debug>

```

(End of definition for \tag_struct_use_num:n. This function is documented on page 95.)

\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.

```

1046 (*package)
1047 \cs_new:Npn \tag_struct_object_ref:n #1
1048 {
1049   \pdf_object_ref:n {__tag/struct/#1}
1050 }
1051 \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 95.)

\tag_struct_gput:nnn This is a command that allows to update the data of a structure. This often can't 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)

```

1052 \cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1053 {
1054   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1055   { %warning??
1056     \use_none:nn
1057   }
1058   {#1}{#3}
1059 }
1060 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1061 </package>

```

(End of definition for \tag_struct_gput:nnn. This function is documented on page 96.)

__tag_struct_gput_data_ref:nn

```

1062 (*package)
1063 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1064   % #1 receiving struct num, #2 list of object ref
1065   {
1066     \prop_get:cnN
1067     { g__tag_struct_#1_prop }
1068     {Ref}
1069     \l__tag_get_tmpc_tl
1070     \__tag_struct_prop_gput:nne
1071     { #1 }
1072     { Ref }
1073     { \quark_if_no_value:NF\l__tag_get_tmpc_tl { \l__tag_get_tmpc_tl\c_space_tl }#2 }
1074   }
1075 \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`
`\tag_struct_parent_int:`

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

It must be used together with `\tag_struct_parent_int:` to insert an annotation. TODO: decide how it should be guarded if tagging is deactivated.

```

1076 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1077                                     %#2 struct parent num
1078 {
1079   \__tag_check_if_active_struct:T
1080   {
1081     \__tag_struct_insert_annot:nn {#1}{#2}
1082   }
1083 }
1084
1085 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1086 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1087
1088 \</package>
1089

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:`. These functions are documented on page 95.)

7 Attributes and attribute classes

```

1090 \<header>
1091 \ProvidesExplPackage {tagpdf-attr-code} {2023-12-18} {0.98r}
1092 {part of tagpdf - code related to attributes and attribute classes}
1093 \</header>

```

7.1 Variables

`\g__tag_attr_entries_prop`
`\g__tag_attr_class_used_seq`
`\g__tag_attr_objref_prop`
`\l__tag_attr_value_tl`

`\g__@@_attr_entries_prop` will store attribute names and their dictionary content.

`\g__@@_attr_class_used_seq` will hold the attributes which have been used as class name. `\l__@@_attr_value_tl` is used to build the attribute array or key. Everytime an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in `\g__@@_attr_objref_prop`

```

1094 \<package>
1095 \prop_new:N \g__tag_attr_entries_prop
1096 \seq_new:N \g__tag_attr_class_used_seq
1097 \tl_new:N \l__tag_attr_value_tl
1098 \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`
`newattribute_␣(setup-key)`

This allows to define attributes. Defined attributes are stored in a global property. `newattribute` expects two brace group, the name and the content. The content typically needs an `/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
{
  newattribute =
    {TH-col}{/O /Table /Scope /Column},
  newattribute =
    {TH-row}{/O /Table /Scope /Row},
}

1099 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1100 {
1101   \prop_gput:Nen \g__tag_attr_entries_prop
1102   {\pdf_name_from_unicode_e:n{#1}}{#2}
1103 }
1104
1105 \keys_define:nn { __tag / setup }
1106 {
1107   newattribute .code:n =
1108   {
1109     \__tag_attr_new_entry:nn #1
1110   }
1111 }

```

(End of definition for `__tag_attr_new_entry:nn` and `newattribute` (setup-key). This function is documented on page 98.)

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

```

1112 \keys_define:nn { __tag / struct }
1113 {
1114   attribute-class .code:n =
1115   {
1116     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1117     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1118     we convert the names into pdf names with slash
1119     \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1120     {
1121       \pdf_name_from_unicode_e:n {##1}
1122     }
1123     \seq_map_inline:Nn \l__tag_tmpa_seq
1124     {
1125       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1126       {
1127         \msg_error:nnn { tag } { attr-unknown } { ##1 }
1128       }
1129       \seq_gput_left:Nn \g__tag_attr_class_used_seq { ##1 }
1130     }
1131     \tl_set:Ne \l__tag_tmpa_tl
1132     {
1133       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1134       \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1135       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1136     }
1137     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1138     {

```

```

1138     \__tag_struct_prop_gput:nne
1139     { \int_use:N \c@g__tag_struct_abs_int }
1140     { C }
1141     { \l__tag_tmpa_tl }
1142     %\prop_show:c { g__tag_struct \int_eval:n {\c@g__tag_struct_abs_int}_prop }
1143   }
1144 }
1145 }

```

(End of definition for attribute-class (struct-key). This function is documented on page 98.)

attribute_␣(struct-key)

```

1146 \keys_define:nn { __tag / struct }
1147 {
1148   attribute .code:n = % A property (attribute, value currently a dictionary)
1149   {
1150     \clist_set:Ne          \l__tag_tmpa_clist { #1 }
1151     \clist_if_empty:NF \l__tag_tmpa_clist
1152     {
1153       \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1154       we convert the names into pdf names with slash
1155       \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1156       {
1157         \pdf_name_from_unicode_e:n {##1}
1158       }
1159       \tl_set:Ne \l__tag_attr_value_tl
1160       {
1161         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%
1162       }
1163       \seq_map_inline:Nn \l__tag_tmpa_seq
1164       {
1165         \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1166         {
1167           \msg_error:nnn { tag } { attr-unknown } { ##1 }
1168         }
1169         \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
1170         {%\prop_show:N \g__tag_attr_entries_prop
1171          \pdf_object_unnamed_write:ne
1172          { dict }
1173          {
1174            \prop_item:Nn \g__tag_attr_entries_prop {##1}
1175          }
1176          \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1177        }
1178        \tl_put_right:Ne \l__tag_attr_value_tl
1179        {
1180          \c_space_tl
1181          \prop_item:Nn \g__tag_attr_objref_prop {##1}
1182        }
1183        % \tl_show:N \l__tag_attr_value_tl
1184      }
1185      \tl_put_right:Ne \l__tag_attr_value_tl
1186      { %[
1187        \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}%

```

```

1187         }
1188     % \tl_show:N \l__tag_attr_value_tl
1189     \__tag_struct_prop_gput:nne
1190     { \int_use:N \c@g__tag_struct_abs_int }
1191     { A }
1192     { \l__tag_attr_value_tl }
1193 }
1194 },
1195 }
1196 \end{package}

```

(End of definition for attribute (struct-key). This function is documented on page 98.)

Part VIII

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2023-12-18} {0.98r}
4 {tagpdf~driver~for~luatex}
```

1 Loading the lua

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

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

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

```

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

```

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

```

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

62 \langle /luatex \rangle

The module declaration

```

63  $\langle$ *lua $\rangle$ 
64 -- tagpdf.lua
65 -- Ulrike Fischer
66
67 local ProvidesLuaModule = {
68   name      = "tagpdf",
69   version   = "0.98r",      --TAGVERSION
70   date      = "2023-12-18", --TAGDATE
71   description = "tagpdf lua code",
72   license    = "The LATEX Project Public License 1.3c"
73 }
74
75 if luatexbase and luatexbase.provides_module then
76   luatexbase.provides_module (ProvidesLuaModule)
77 end
78
79 --[[

```

```

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

```

Some comments about the lua structure.

```

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

```

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

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

```

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

```

with this token we can query the state of the boolean and so detect if unmarked nodes should be marked as attributes

```

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

```

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

```

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

```

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

```

165 ltx          = ltx          or { }
166 ltx.__tag    = ltx.__tag    or { }
167 ltx.__tag.mc  = ltx.__tag.mc or { } -- mc data
168 ltx.__tag.struc = ltx.__tag.struc or { } -- struct data
169 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
170                                     -- wasn't a so great idea ...

```

```

171                                     -- g__tag_role_tags_seq used by tag<-> is in this table
172                                     -- used for pure lua tables too now!
173 ltx.__tag.page      = ltx.__tag.page   or { } -- page data, currently only i->{0->mcnum,1->mcnum}
174 ltx.__tag.trace     = ltx.__tag.trace  or { } -- show commands
175 ltx.__tag.func       = ltx.__tag.func   or { } -- functions
176 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
`ltx.__tag.trace.log` will output the message to the log/terminal if the current loglevel is greater or equal than
num.

```

177 local __tag_log =
178   function (message,loglevel)
179     if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
180       texio.write_nl("tagpdf: ".. message)
181     end
182   end
183
184 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.

```

185 function ltx.__tag.trace.show_seq (seq)
186   if (type(seq) == "table") then
187     for i,v in ipairs(seq) do
188       __tag_log ("[" .. i .. "] => " .. tostring(v),1)
189     end
190   else
191     __tag_log ("sequence " .. tostring(seq) .. " not found",1)
192   end
193 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
`ltx.__tag.trace.show_prop` `\@@_prop_show:N` function.

```

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

```

```

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

```

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

```

234 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
235 for i = min, max do
236   ltx.__tag.trace.show_mc_data (i,loglevel)
237 end
238 texio.write_nl("")
239 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.

```

240 function ltx.__tag.trace.show_struct_data (num)
241 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
242   for k,v in ipairs(ltx.__tag.struct[num]) do
243     __tag_log ("struct "..num..": " ..tostring(k)..=>" ..tostring(v),1)
244   end
245 else
246   __tag_log ("struct "..num.." not found ",1)
247 end
248 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).

```

249 local __tag_get_mc_cnt_type_tag = function (n)
250   local mcnt      = nodegetattribute(n,mcntattributeid) or -1
251   local mctype     = nodegetattribute(n,mctypeattributeid) or -1
252   local tag        = ltx.__tag.func.get_tag_from(mctype)
253   return mcnt,mctype,tag
254 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.

```

255 local function __tag_get_mathsubtype (mathnode)
256   if mathnode.subtype == 0 then
257     subtype = "beginmath"
258   else
259     subtype = "endmath"
260   end
261   return subtype
262 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.

```

263 ltx.__tag.tables.role_tag_attribute = {}
264 ltx.__tag.tables.role_attribute_tag = {}

```

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

`ltx.__tag.func.alloctag`

```

265 local __tag_alloctag =
266   function (tag)
267     if not ltx.__tag.tables.role_tag_attribute[tag] then
268       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
269       ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
270       __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
271     end
272   end
273 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.

```

274 local __tag_get_num_from =
275 function (tag)
276   if ltx.__tag.tables.role_tag_attribute[tag] then
277     a= ltx.__tag.tables.role_tag_attribute[tag]
278   else
279     a= -1
280   end
281   return a
282 end
283
284 ltx.__tag.func.get_num_from = __tag_get_num_from
285
286 function ltx.__tag.func.output_num_from (tag)
287   local num = __tag_get_num_from (tag)
288   tex.sprint(catlatex,num)
289   if num == -1 then
290     __tag_log ("Unknown tag "..tag.." used")
291   end
292 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.

```

293 local __tag_get_tag_from =
294 function (num)
295   if ltx.__tag.tables.role_attribute_tag[num] then
296     a = ltx.__tag.tables.role_attribute_tag[num]
297   else
298     a= "UNKNOWN"
299   end
300   return a
301 end
302
303 ltx.__tag.func.get_tag_from = __tag_get_tag_from
304
305 function ltx.__tag.func.output_tag_from (num)
306   tex.sprint(catlatex,__tag_get_tag_from (num))
307 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.

```

308 function ltx.__tag.func.store_mc_data (num,key,data)
309   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
310   ltx.__tag.mc[num][key] = data
311   __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
312 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.

```

313 function ltx.__tag.func.store_mc_label (label,num)
314   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
315   ltx.__tag.mc.labels[label] = num
316 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.

```

317 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
318   ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
319   ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
320   local kidtable = {kid=kid,page=page}
321   tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
322 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.

```

323 function ltx.__tag.func.mc_num_of_kids (mcnum)
324   local num = 0
325   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
326     num = #ltx.__tag.mc[mcnum]["kids"]
327   end
328   ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
329   return num
330 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

```

331 local __tag_backend_create_emc_node
332 if tex.outputmode == 0 then
333   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
334     function __tag_backend_create_emc_node ()
335       local emcnode = nodenew("whatsit","special")
336       emcnode.data = "pdf:code EMC"
337       return emcnode
338     end
339   else -- assume a dvips variant
340     function __tag_backend_create_emc_node ()
341       local emcnode = nodenew("whatsit","special")
342       emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
343       return emcnode
344     end
345   end
346 else -- pdf mode

```

```

347 function __tag_backend_create_emc_node ()
348     local emcnode = nodenew("whatsit","pdf_literal")
349     emcnode.data = "EMC"
350     emcnode.mode=1
351     return emcnode
352 end
353 end
354
355 local function __tag_insert_emc_node (head,current)
356     local emcnode= __tag_backend_create_emc_node()
357     head = node.insert_before(head,current,emcnode)
358     return head
359 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
360 local __tag_backend_create_bmc_node
361 if tex.outputmode == 0 then
362     if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
363         function __tag_backend_create_bmc_node (tag)
364             local bmcnode = nodenew("whatsit","special")
365             bmcnode.data = "pdf:code /"..tag.." BMC"
366             return bmcnode
367         end
368     else -- assume a dvips variant
369         function __tag_backend_create_bmc_node (tag)
370             local bmcnode = nodenew("whatsit","special")
371             bmcnode.data = "ps:SDict begin mark/"..tag.." BMC pdfmark end"
372             return bmcnode
373         end
374     end
375 else -- pdf mode
376     function __tag_backend_create_bmc_node (tag)
377         local bmcnode = nodenew("whatsit","pdf_literal")
378         bmcnode.data = "/"..tag.." BMC"
379         bmcnode.mode=1
380         return bmcnode
381     end
382 end
383
384 local function __tag_insert_bmc_node (head,current,tag)
385     local bmcnode = __tag_backend_create_bmc_node (tag)
386     head = node.insert_before(head,current,bmcnode)
387     return head
388 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.
389 local __tag_backend_create_bdc_node
390
391 if tex.outputmode == 0 then

```

```

392 if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
393   function __tag_backend_create_bdc_node (tag,dict)
394     local bdcnode = nodenew("whatsit","special")
395     bdcnode.data = "pdf:code /"..tag.."<<..dict..">> BDC"
396     return bdcnode
397   end
398 else -- assume a dvips variant
399   function __tag_backend_create_bdc_node (tag,dict)
400     local bdcnode = nodenew("whatsit","special")
401     bdcnode.data = "ps:SDict begin mark/"..tag.."<<..dict..">> BDC pdfmark end"
402     return bdcnode
403   end
404 end
405 else -- pdf mode
406   function __tag_backend_create_bdc_node (tag,dict)
407     local bdcnode = nodenew("whatsit","pdf_literal")
408     bdcnode.data = "/"..tag.."<<..dict..">> BDC"
409     bdcnode.mode=1
410     return bdcnode
411   end
412 end
413
414 local function __tag_insert_bdc_node (head,current,tag,dict)
415   bdcnode= __tag_backend_create_bdc_node (tag,dict)
416   head = node.insert_before(head,current,bdcnode)
417   return head
418 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

`ltx.__tag.func.pdf_object_ref`

```

419 local function __tag_pdf_object_ref (name)
420   local tokenname = 'c__pdf_backend_object_'..name..'__int'
421   local object = token.create(tokenname).index..' 0 R'
422   return object
423 end
424 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.

```

425 local function __tag_show_spacemark (head,current,color,height)
426   local markcolor = color or "1 0 0"
427   local markheight = height or 10
428   local pdfstring
429   if tex.outputmode == 0 then
430     -- ignore dvi mode for now
431   else

```

```

432 pdfstring = node.new("whatsit","pdf_literal")
433 pdfstring.data =
434 string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
3,markheight)
435 head = node.insert_after(head,current,pdfstring)
436 return head
437 end
438 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 439 local function __tag_fakespace()
440 tex.setattribute(iwspaceattributeid,1)
441 tex.setattribute(iwfontattributeid,font.current())
442 end
443 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.

```

444 --[[ a function to mark up places where real space chars should be inserted
445 it only sets an attribute.
446 --]]
447
448 local function __tag_mark_spaces (head)
449 local inside_math = false
450 for n in nodetraverse(head) do
451 local id = n.id
452 if id == GLYPH then
453 local glyph = n
454 if glyph.next and (glyph.next.id == GLUE)
455 and not inside_math and (glyph.next.width > 0)
456 then
457 nodesetattribute(glyph.next,iwspaceattributeid,1)
458 nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
459 -- for debugging
460 if ltx.__tag.trace.showspace then
461 __tag_show_spacemark (head,glyph)
462 end
463 elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
464 local kern = glyph.next
465 if kern.next and (kern.next.id== GLUE) and (kern.next.width > 0)
466 then
467 nodesetattribute(kern.next,iwspaceattributeid,1)
468 nodesetattribute(kern.next,iwfontattributeid,glyph.font)
469 end
470 end
471 -- look also back
472 if glyph.prev and (glyph.prev.id == GLUE)
473 and not inside_math
474 and (glyph.prev.width > 0)

```

```

475         and not nodehasattribute(glyph.prev,iwspaceattributeid)
476     then
477         nodesetattribute(glyph.prev,iwspaceattributeid,1)
478         nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
479     -- for debugging
480     if ltx.__tag.trace.showspace then
481         __tag_show_spacemark (head,glyph)
482     end
483 end
484 elseif id == PENALTY then
485     local glyph = n
486     -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
487     if glyph.next and (glyph.next.id == GLUE)
488     and not inside_math and (glyph.next.width > 0) and n.subtype==0
489     then
490         nodesetattribute(glyph.next,iwspaceattributeid,1)
491         -- nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
492     -- for debugging
493     if ltx.__tag.trace.showspace then
494         __tag_show_spacemark (head,glyph)
495     end
496     end
497 elseif id == MATH then
498     inside_math = (n.subtype == 0)
499 end
500 end
501 return head
502 end

```

(End of definition for __tag_mark_spaces.)

```

__tag_activate_mark_space
ltx.__tag.func.markspaceon
ltx.__tag.func.markspaceoff

```

Theses functions add/remove the function which marks the spaces to the callbacks **pre_linebreak_filter** and **hpack_filter**

```

503 local function __tag_activate_mark_space ()
504 if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
505     luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
506     luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
507 end
508 end
509
510 ltx.__tag.func.markspaceon=__tag_activate_mark_space
511
512 local function __tag_deactivate_mark_space ()
513 if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
514     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
515     luatexbase.remove_from_callback("hpack_filter","markspaces")
516 end
517 end
518
519 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.

```

520 local default_space_char = nodenew(GLYPH)

```

```

521 local default_fontid      = fontid("TU/lmr/m/n/10")
522 default_space_char.char   = 32
523 default_space_char.font   = default_fontid

```

And a function to check as best as possible if a font has a space:

```

524 local function __tag_font_has_space (fontid)
525   t= fonts.hashes.identifiers[fontid]
526   if luaotfload.aux.slot_of_name(fontid,"space")
527     or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
528   then
529     return true
530   else
531     return false
532   end
533 end

```

```

__tag_space_chars_shipout
ltx.__tag.func.space_chars_shipout

```

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

```

534 local function __tag_space_chars_shipout (box)
535   local head = box.head
536   if head then
537     for n in node.traverse(head) do
538       local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
539       if n.id == HLIST then -- enter the hlist
540         __tag_space_chars_shipout (n)
541       elseif n.id == VLIST then -- enter the vlist
542         __tag_space_chars_shipout (n)
543       elseif n.id == GLUE then
544         if ltx.__tag.trace.showspace and spaceattr==1 then
545           __tag_show_spacemark (head,n,"0 1 0")
546         end
547         if spaceattr==1 then
548           local space
549           local space_char = node.copy(default_space_char)
550           local curfont = nodegetattribute(n,iwfontattributeid)
551           ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
552           if curfont and
553             -- luaotfload.aux.slot_of_name(curfont,"space")
554             __tag_font_has_space (curfont)
555           then
556             space_char.font=curfont
557           end
558           head, space = node.insert_before(head, n, space_char) --
559           n.width      = n.width - space.width
560           space.attr   = n.attr
561         end
562       end
563     end
564     box.head = head
565   end
566 end
567
568 function ltx.__tag.func.space_chars_shipout (box)

```

```

569 __tag_space_chars_shipout (box)
570 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.

```

571 function ltx.__tag.func.mc_insert_kids (mcnum,single)
572   if ltx.__tag.mc[mcnum] then
573     ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
574     if ltx.__tag.mc[mcnum]["kids"] then
575       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
576         tex.sprint("(")
577       end
578       for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
579         local kidnum = kidstable["kid"]
580         local kidpage = kidstable["page"]
581         local kidpageobjnum = pdfpageref(kidpage)
582         ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
583           " insert KID " .. i..
584           " with num " .. kidnum ..
585           " on page " .. kidpage.."/"..kidpageobjnum,3)
586         tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> " )
587       end
588       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
589         tex.sprint("]")
590       end
591     else
592       -- this is typically not a problem, e.g. empty hbox in footer/header can
593       -- trigger this warning.
594       ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
595       if single==1 then
596         tex.sprint("null")
597       end
598     end
599   else
600     ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
601   end
602 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.

```

603 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
604   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
605   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
606   -- a structure can contain more than on mc chunk, the content should be ordered
607   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)

```

```

608 ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
609                     mcnum.." inserted in struct "..structnum,3)
610 -- but every mc can only be in one structure
611 ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
612 ltx.__tag.mc[mcnum]["parent"] = structnum
613 end
614

```

(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.

```

615 -- pay attention: lua counts arrays from 1, tex pages from one
616 -- mcid and arrays in pdf count from 0.
617 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagencnt,page)
618 ltx.__tag.page[page] = ltx.__tag.page[page] or {}
619 ltx.__tag.page[page][mcpagencnt] = mcnum
620 ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
621                     ": inserting MCID " .. mcpagencnt .. " => " .. mcnum,3)
622 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)

```

623 local function __tag_update_mc_attributes (head,mcnum,type)
624 for n in node.traverse(head) do
625     node.set_attribute(n,mccntattributeid,mcnum)
626     node.set_attribute(n,mctypeattributeid,type)
627     if n.id == HLIST or n.id == VLIST then
628         __tag_update_mc_attributes (n.list,mcnum,type)
629     end
630 end
631 return head
632 end
633 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.

```

634 --[[
635     Now follows the core function
636     It wades through the shipout box and checks the attributes
637     ARGUMENTS
638     box: is a box,
639     mcpagencnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
640     mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a c
641     mcpopen: num, records if some bdc/emc is open
642     These arguments are only needed for log messages, if not present are replaces by fix stri
643     name: string to describe the box
644     mctypeprev: num, the type attribute of the previous node/whatever
645
646     there are lots of logging messages currently. Should be cleaned up in due course.

```



```

647     One should also find ways to make the function shorter.
648 --]]
649
650 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
651     local name = name or ("SOMEBOX")
652     local mctypeprev = mctypeprev or -1
653     local abspage = status.total_pages + 1 -- the real counter is increased
654                                           -- inside the box so one off
655                                           -- if the callback is not used. (???)
656     ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
657     ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
658                         " prev "..mccntprev ..
659                         " type prev "..mctypeprev,4)
660     ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
661                         " TYPE ".. node.type(node.getid(box)),3)
662     local head = box.head -- ShipoutBox is a vlist?
663     if head then
664         mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
665         ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
666                             node.type(node.getid(head))..
667                             " MC"..tostring(mccnthead)..
668                             " => TAG " .. tostring(mctypehead)..
669                             " => ".. tostring(taghead),3)
670     else
671         ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
672                             tostring(head),3)
673     end
674     for n in node.traverse(head) do
675         local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
676         local spaceattr = node.getattribute(n,iwspaceattributeid) or -1
677         ltx.__tag.trace.log ("INFO TAG-NODE: "..
678                             node.type(node.getid(n))..
679                             " MC".. tostring(mccnt)..
680                             " => TAG ".. tostring(mctype)..
681                             " => " .. tostring(tag),3)
682         if n.id == HLIST
683         then -- enter the hlist
684             mcopen,mcpagecnt,mccntprev,mctypeprev=
685                 ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypeprev)
686         elseif n.id == VLIST then -- enter the vlist
687             mcopen,mcpagecnt,mccntprev,mctypeprev=
688                 ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypeprev)
689         elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but tl
690                                                     -- been done if the previous shipout wandering, so here it
691         elseif n.id == LOCAL_PAR then -- local_par is ignored
692         elseif n.id == PENALTY then -- penalty is ignored
693         elseif n.id == KERN then -- kern is ignored
694             ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
695                                 node.type(node.getid(n)).." "..n.subtype,4)
696         else
697             -- math is currently only logged.
698             -- we could mark the whole as math
699             -- for inner processing the mlist_to_hlist callback is probably needed.
700             if n.id == MATH then

```

```

701     ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
702         node.type(node.getid(n)).." " ..__tag_get_mathsubtype(n),4)
703 end
704 -- endmath
705 ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
706     mccnt.." prev "..mccntprev,4)
707 if mccnt~=mccntprev then -- a new mc chunk
708     ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
709         node.type(node.getid(n))..
710         " MC"..tostring(mccnt)..
711         " <=> PREVIOUS "..tostring(mccntprev),4)
712 if mcpopen~=0 then -- there is a chunk open, close it (hope there is only one ...
713     box.list=__tag_insert_emc_node (box.list,n)
714     mcpopen = mcpopen - 1
715     ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
716         mcpagecnt .. " MCOPEX = " .. mcpopen,3)
717 if mcpopen ~=0 then
718     ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcpopen,1)
719 end
720 end
721 if ltx.__tag.mc[mccnt] then
722     if ltx.__tag.mc[mccnt]["artifact"] then
723         ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
724             tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
725     if ltx.__tag.mc[mccnt]["artifact"] == "" then
726         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
727     else
728         box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]
729     end
730 else
731     ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
732         tostring(tag),3)
733     mcpagecnt = mcpagecnt +1
734     ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
735     local dict= "/MCID "..mcpagecnt
736     if ltx.__tag.mc[mccnt]["raw"] then
737         ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
738             tostring(ltx.__tag.mc[mccnt]["raw"]),3)
739         dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
740     end
741     if ltx.__tag.mc[mccnt]["alt"] then
742         ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
743             tostring(ltx.__tag.mc[mccnt]["alt"]),3)
744         dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
745     end
746     if ltx.__tag.mc[mccnt]["actualtext"] then
747         ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
748             tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
749         dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
750     end
751     box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
752     ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
753     ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
754     ltx.__tag.trace.show_mc_data (mccnt,3)

```

```

755     end
756     mcopen = mcopen + 1
757   else
758     if tagunmarkedbool.mode == truebool.mode then
759       ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
760       box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
761       mcopen = mcopen + 1
762     else
763       ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
764     end
765   end
766   mccntprev = mccnt
767 end
768 end -- end if
769 end -- end for
770 if head then
771   mccnthead, mctypehead, taghead = __tag_get_mc_cnt_type_tag (head)
772   ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
773                       node.type(node.getid(head))..
774                       " MC"..tostring(mccnthead)..
775                       " => TAG "..tostring(mctypehead)..
776                       " => "..tostring(taghead),4)
777 else
778   ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
779 end
780 ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
781                     tostring(name)..
782                     " TYPE ".. node.type(node.getid(box)),4)
783 return mcopen,mcpagecnt,mccntprev,mctypeprev
784 end
785

```

(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.

```

786 function ltx.__tag.func.mark_shipout (box)
787   mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
788   if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
789     local emcnode = __tag_backend_create_emc_node ()
790     local list = box.list
791     if list then
792       list = node.insert_after (list,node.tail(list),emcnode)
793       mcopen = mcopen - 1
794       ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
795     else
796       ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
797     end
798     if mcopen ~=0 then
799       ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
800     end
801   end
802 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.

```
803 function ltx.__tag.func.fill_parent_tree_line (page)
804     -- we need to get page-> i=kid -> mcnum -> structnum
805     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
806     local numsentry = ""
807     local pdfpage = page-1
808     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
809         mcchunks=#ltx.__tag.page[page]
810         ltx.__tag.trace.log("INFO PARENTTREE-NUM:  page "..
811             page.." has "..mcchunks.." +1 Elements ",4)
812         for i=0,mcchunks do
813             -- what does this log??
814             ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS:  "..
815                 ltx.__tag.page[page][i],4)
816         end
817         if mcchunks == 0 then
818             -- only one chunk so no need for an array
819             local mcnum = ltx.__tag.page[page][0]
820             local structnum = ltx.__tag.mc[mcnum]["parent"]
821             local propname = "g__tag_struct"..structnum.."_prop"
822             --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
823             local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
824             ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF:  =====>"..
825                 tostring(objref),5)
826             numsentry = pdfpage .. " [".. objref .. "]"
827             ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY:  page " ..
828                 page.. " num entry = ".. numsentry,3)
829         else
830             numsentry = pdfpage .. " ["
831             for i=0,mcchunks do
832                 local mcnum = ltx.__tag.page[page][i]
833                 local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
834                 local propname = "g__tag_struct"..structnum.."_prop"
835                 --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
836                 local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
837                 numsentry = numsentry .. " " .. objref
838             end
839             numsentry = numsentry .. "]"
840             ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY:  page " ..
841                 page.. " num entry = ".. numsentry,3)
842         end
843     else
844         ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA:  page "..page,3)
845     end
846     return numsentry
847 end
848
849 function ltx.__tag.func.output_parenttree (abspage)
```

```

850 for i=1,abspage do
851   line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
852   tex.sprint(catlatex,line)
853 end
854 end

(End of definition for ltx.__tag.func.fill_parent_tree_line and ltx.__tag.func.output_parenttree.)
855 </lua>

```

Part IX

The tagpdf-roles module

Tags, roles and namespace code

Part of the tagpdf package

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

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

The key-value list knows the following keys:

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

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

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

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

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2023-12-18} {0.98r}
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>

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

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

9 \prop_new:N \g__tag_role_NS_prop

(End of definition for \g__tag_role_NS_prop.)

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

10 \prop_new:N \g__tag_role_index_prop

(End of definition for \g__tag_role_index_prop.)

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

11 \prop_new:N \l__tag_role_debug_prop

(End of definition for \l__tag_role_debug_prop.)

We need also a bunch of temporary variables.

`\l__tag_role_tag_tmpa_tl`

`\l__tag_role_tag_namespace_tmpa_tl`

12 \tl_new:N \l__tag_role_tag_tmpa_tl

`\l__tag_role_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

`\l__tag_role_role_namespace_tmpa_tl`

17 \seq_new:N \l__tag_role_role_namespace_tmpa_tl

(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 currently only loaded for pdf 2.0.

```

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

```

1.3 Adding a new tag

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

__tag_role_alloctag:nnn

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

```

84 \pdf_version_compare:NnTF < {2.0}
85   {

```

```

86 \sys_if_engine luatex:TF
87 {
88   \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 % #1 tagname, ns, type
89   {
90     \lua_now:e { ltx.__tag.func.alloctag ('#1') }
91     \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
92     \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
93     \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
94     \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
95   }
96 }
97 {
98   \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
99   {
100     \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
101     \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
102     \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
103     \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
104   }
105 }
106 }
107 {
108   \sys_if_engine luatex:TF
109   {
110     \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 % #1 tagname, ns, type
111     {
112       \lua_now:e { ltx.__tag.func.alloctag ('#1') }
113       \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
114       \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
115       \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
116       \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
117     }
118   }
119   {
120     \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
121     {
122       \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
123       \prop_gput:cnn {g__tag_role_NS_#2_prop} {#1}{{}}{}
124       \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
125       \prop_gput:cnn {g__tag_role_NS_#2_class_prop} {#1}{#3}
126     }
127   }
128 }
129 \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.

```

130 \cs_new_protected:Nn \__tag_role_add_tag:nn % (new) name, reference to old
131 {

```

checks and messages

```

132   \__tag_check_add_tag_role:nn {#1}{#2}
133   \prop_if_in:NnF \g__tag_role_tags_NS_prop {#1}
134   {
135     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
136     {
137       \msg_info:nnn { tag }{new-tag}{#1}
138     }
139   }

```

now the addition

```

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

```

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

```

146   \tl_if_empty:nF { #2 }
147   {
148     \prop_get:NnN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
149     \quark_if_no_value:NTF \l__tag_tmpa_tl
150     {
151       \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
152     }
153     {
154       \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
155     }
156   }
157 }
158 \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

```

159 \pdf_version_compare:NnT < {2.0}
160 {
161   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 {%#1 tag, #2 NS, #3 tlvar which hold the role tag
162   {
163     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
164     {
165       \tl_set:Nn #3 {#1}
166     }
167     \tl_set:Nn #4 {}
168   }
169   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
170 }
171

```

(End of definition for __tag_role_get:nnNN.)

1.3.2 The pdf 2.0 version

```

172 \cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
173 {
174   \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
175   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
176   {
177     \msg_info:nnn { tag }{new-tag}{#1}
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   \__tag_role_alloctag:nnV {#1}{#2}\l__tag_tmpa_tl

```

Do not remap standard tags. TODO add warning?

```

185   \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
186   {
187     \pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
188     {
189       [
190         \pdf_name_from_unicode_e:n{#3}
191         \c_space_tl
192         \pdf_object_ref:n {tag/NS/#4}
193       ]
194     }
195   }

```

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

```

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

```

(End of definition for __tag_role_add_tag:nnnn.)

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

```

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

```

```

213     %1 tag, #2 NS,
214     %3 tlvar which hold the role tag
215     %4 tlvar which hold the name of the target NS
216   {
217     \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_tmpa_tl
218     {
219       \tl_set:Ne #3 {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
220       \tl_set:Ne #4 {\exp_last_unbraced:NV\use_ii:nn \l__tag_tmpa_tl}
221     }
222     {
223       \tl_set:Nn #3 {#1}
224       \tl_set:Nn #4 {#2}
225     }
226   }
227   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
228 }

```

(End of definition for __tag_role_get:nnNN.)

1.4 Helper command to read the data from files

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

__tag_role_read_namespace_line:nw

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

```

229 \bool_new:N\l__tag_role_update_bool
230 \bool_set_true:N \l__tag_role_update_bool
231 \pdf_version_compare:NnTF < {2.0}
232 {
233   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
234   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
235   {
236     \tl_if_empty:nF { #2 }
237     {
238       \bool_if:NTF \l__tag_role_update_bool
239       {
240         \tl_if_empty:nTF {#5}
241         {
242           \prop_get:NnN \g__tag_role_tags_class_prop {#3}\l__tag_tmpa_tl
243           \quark_if_no_value:NT \l__tag_tmpa_tl
244           {
245             \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
246           }
247         }
248         {
249           \tl_set:Nn \l__tag_tmpa_tl {#5}
250         }
251       }
252       \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
253       \tl_if_eq:nnF {#2}{#3}
254       {
255         \__tag_role_add_tag:nn {#2}{#3}
256       }
257     }
258   }

```

```

256     \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{{#3}{}}
257   }
258   {
259     \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{{#3}{}}
260     \prop_gput:cnn {g__tag_role_NS_#1_class_prop} {#2}{--UNUSED--}
261   }
262 }
263 }
264 }
265 {
266   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
267   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
268   {
269     \tl_if_empty:nF {#2}
270     {
271       \tl_if_empty:nTF {#5}
272       {
273         \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
274         \quark_if_no_value:NT \l__tag_tmpa_tl
275         {
276           \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
277         }
278       }
279       {
280         \tl_set:Nn \l__tag_tmpa_tl {#5}
281       }
282       \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
283       \bool_lazy_and:nnT
284       { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
285       {
286         \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
287       }
288       \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{{#3}{#4}}
289     }
290   }
291 }

```

(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

```

292 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
293 {
294   \prop_if_exist:cF {g__tag_role_NS_#1_prop}
295   { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
296   \file_if_exist:nTF { tagpdf-ns-#2.def }
297   {
298     \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
299     \msg_info:nnn {tag}{read-namespace}{#2}
300     \ior_map_inline:Nn \g_tmpa_ior
301     {
302       \__tag_role_read_namespace_line:nw {#1} ##1,,,\q_stop
303     }
304     \ior_close:N\g_tmpa_ior
305   }

```

```

306     {
307       \msg_info:nnn{tag}{namespace-missing}{#2}
308     }
309   }
310

```

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

```

\__tag_role_read_namespace:n This command reads the default namespace file.
311 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
312 {
313   \__tag_role_read_namespace:nn {#1}{#1}
314 }

```

(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.

```

315 \__tag_role_read_namespace:n {pdf}
316 \__tag_role_read_namespace:n {pdf2}
317 \pdf_version_compare:NnF < {2.0}
318   {\__tag_role_read_namespace:n {mathml}}

```

in pdf 1.7 the following namespaces should only store the settings for later use:

```

319 \bool_set_false:N\l__tag_role_update_bool
320 \__tag_role_read_namespace:n {latex-inline}
321 \__tag_role_read_namespace:n {latex-book}
322 \bool_set_true:N\l__tag_role_update_bool
323 \__tag_role_read_namespace:n {latex}
324 \__tag_role_read_namespace:nn {latex} {latex-lab}
325 \__tag_role_read_namespace:n {pdf}
326 \__tag_role_read_namespace:n {pdf2}

```

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

```

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

```



```

346         {
347         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ latex-book }
348         }
349     }
350 }
351 }

```

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.

```

352 \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` `\c_tag_role_rules_num_prop` These two properties map the rule strings to numbers and back. There are in tagpdf-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!

```

353 \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
354 {
355     \intarray_gset:Nnn \g__tag_role_parent_child_intarray
356     { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
357 }

```

(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

```

358 \int_zero:N \l__tag_tmpa_int

```

Open the file depending on the PDF version

```

359 \pdf_version_compare:NnTF < {2.0}
360 {
361     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
362 }
363 {
364     \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
365 }

```

Now the main loop over the file

```

366 \ior_map_inline:Nn \g_tmpa_ior
367 {

```

ignore lines containing only comments

```

368     \tl_if_empty:nF{#1}
369     {

```

count the lines ...

```
370 \int_incr:N\l__tag_tmpa_int
```

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

```
371 \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
```

```
372 \int_compare:nNnTF {\l__tag_tmpa_int}=1
```

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

```
373 {
374   \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
375   {
376     \prop_gput:Nne\g__tag_role_index_prop
377     {##2}
378     {\int_compare:nNnT{##1}<{10}{0}{##1}}
379   }
380 }
```

now the data lines.

```
381 {
382   \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
```

get the name of the child tag from the first column

```
383 \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

get the number of the child, and store it in \l__tag_tmppb_tl

```
384 \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmppb_tl
```

remove column 2+3

```
385 \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
386 \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.

```
387 \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
388 {
389   \exp_args:Nne
390   \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmppb_tl}{ ##2 }
391 }
392 }
393 }
394 }
```

close the read handle.

```
395 \ior_close:N\g_tmpa_ior
```

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

```
396 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
397 \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
398 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
399 \pdf_version_compare:NnTF < {2.0}
400 {
401   \int_step_inline:nn{6}
402   {
403     \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
404   }
405 }
406 {
```

```

407 \int_step_inline:nn{10}
408 {
409   \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
410 }
all mathml tags are currently handled identically
411 \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
412 \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
413 \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
414 {
415   \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
416 }
417 \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
418 }

```

1.6.2 Retrieving the parent-child rule

`__tag_role_get_parent_child_rule:nnnN` 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.

```

419 \tl_new:N \l__tag_parent_child_check_tl
420 \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
421 % #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
422 % #4 tl for state
423 {
424   \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
425   \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
426   \bool_lazy_and:nnTF
427     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
428     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
429   {

```

Get the rule from the intarray

```

430 \tl_set:Nc#4
431 {
432   \intarray_item:Nn
433   \g__tag_role_parent_child_intarray
434   {\l__tag_tmpa_tl\l__tag_tmpb_tl}
435 }

```

If the state is something is wrong ...

```

436 \int_compare:nNnT
437 {#4} = {\prop_item:Nn\c__tag_role_rules_prop{}}
438 {
439   %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.

```

440 }

```

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

```

441     \group_begin:
442     \int_compare:nNtT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
443     {
444         \prop_get:NVNF\c__tag_role_rules_num_prop #4 \l__tag_tmpa_tl
445         {
446             \tl_set:Nn \l__tag_tmpa_tl {unknown}
447         }
448         \tl_set:Nn \l__tag_tmpb_tl {#1}
449         \msg_note:nneee
450         { tag }
451         { role-parent-child }
452         { #1 }
453         { #2 }
454         {
455             #4~(=\l__tag_tmpa_tl')
456             \iow_newline:
457             #3
458         }
459     }
460     \group_end:
461 }
462 {
463     \tl_set:Nn#4 {0}
464     \msg_warning:nneee
465     { tag }
466     {role-parent-child}
467     { #1 }
468     { #2 }
469     { unknown! }
470 }
471 }
472 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnnN {VVVN,VVnN}

```

(End of definition for __tag_role_get_parent_child_rule:nnnN.)

__tag_check_parent_child:nnnnN

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

```

473 \pdf_version_compare:NnTF < {2.0}
474 {
475     \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
476     {%#1 parent tag, #2 NS, #3 child tag, #4 NS, #5 tl var
477     {

```

for debugging messages we store the arguments.

```

478         \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
479         \prop_put:Nnn \l__tag_role_debug_prop {child} {#3}

```

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

```

480         \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
481         {

```

```

482     \tl_set:Nn \l__tag_tmpa_tl {#1}
483   }
484   {
485     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
486     {
487       \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
488     }
489   }

```

now the child

```

490     \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
491     {
492       \tl_set:Nn \l__tag_tmpb_tl {#3}
493     }
494     {
495       \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
496       {
497         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
498       }
499     }

```

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

```

500     \bool_lazy_and:nnTF
501     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
502     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
503     {
504       \__tag_role_get_parent_child_rule:VVnN
505       \l__tag_tmpa_tl \l__tag_tmpb_tl
506       {Rolemapped~from:~'#1'~-->~'#3'}
507       #5
508     }
509     {
510       \tl_set:Nn #5 {0}
511       \msg_warning:nneee
512       { tag }
513       {role-parent-child}
514       { #1 }
515       { #3 }
516       { unknown! }
517     }
518   }
519   \cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
520   {
521     \__tag_check_parent_child:nnnnN {#1}{#2}{#3}
522   }
523 }

```

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.

```

524   {
525     \cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
526     {
527       \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
528       \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl

```

```

529     \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
530     \bool_lazy_and:nnTF
531     { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl }
532     { ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl }
533     {
534         \__tag_check_parent_child:nVnVN
535         {#1}\l__tag_role_tag_namespace_tmpa_tl
536         {#2}\l__tag_role_tag_namespace_tmpb_tl
537         #3
538     }
539     {
540         \tl_set:Nn #3 {0}
541         \msg_warning:nneee
542         { tag }
543         {role-parent-child}
544         { #1 }
545         { #2 }
546         { unknown! }
547     }
548 }

```

and now the real command.

```

549     \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS, tl var
550     {
551         \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
552         \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

```

553     \tl_if_empty:nTF {#2}
554     {
555         \tl_set:Nn \l__tag_tmpa_tl {#1}
556     }
557     {
558         \prop_get:cnNTF
559         { g__tag_role_NS_#2_prop }
560         {#1}
561         \l__tag_tmpa_tl
562         {
563             \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
564             \tl_if_empty:NT\l__tag_tmpa_tl
565             {
566                 \tl_set:Nn \l__tag_tmpa_tl {#1}
567             }
568         }
569         {
570             \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
571         }
572     }

```

and the same for the child If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```

573     \tl_if_empty:nTF {#4}
574     {
575         \tl_set:Nn \l__tag_tmpb_tl {#3}

```

```

576     }
577     {
578         \prop_get:cnNTF
579         { g__tag_role_NS_#4_prop }
580         {#3}
581         \l__tag_tmpb_tl
582         {
583             \tl_set:Nc \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
584             \tl_if_empty:NT\l__tag_tmpb_tl
585             {
586                 \tl_set:Nn \l__tag_tmpb_tl {#3}
587             }
588         }
589         {
590             \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
591         }
592     }

```

and now get the relation

```

593     \bool_lazy_and:nnTF
594     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
595     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
596     {
597         \__tag_role_get_parent_child_rule:VVnN
598         \l__tag_tmpa_tl \l__tag_tmpb_tl
599         {Rolemapped~from~'#1/#2'--->~'~'#3\str_if_empty:nF{#4}{/#4}' }
600         #5
601     }
602     {
603         \tl_set:Nn #5 {0}
604         \msg_warning:nneee
605         { tag }
606         {role-parent-child}
607         { #1 }
608         { #3 }
609         { unknown! }
610     }
611 }
612 }
613 \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
614 \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
615 \endpackage

```

(End of definition for __tag_check_parent_child:nnnnN.)

\tag_check_child:nnTF

```

616 (base)\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true:
617 *package}
618 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}
619 {
620     \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
621     \__tag_struct_get_parentrole:eNN
622     {\l__tag_tmpa_tl}
623     \l__tag_get_parent_tmpa_tl
624     \l__tag_get_parent_tmpb_tl

```

```

625 \__tag_check_parent_child:VVnnN
626 \l__tag_get_parent_tmpa_tl
627 \l__tag_get_parent_tmpb_tl
628 {#1}{#2}
629 \l__tag_parent_child_check_tl
630 \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
631 {\prg_return_false:}
632 {\prg_return_true:}
633 }

```

(End of definition for `\tag_check_child:nNnTF`. This function is documented on page 150.)

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
634 \tl_new:N \l__tag_role_remap_tag_tl
635 \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?

```

636 \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.

```

637 \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:

```

(End of definition for `__tag_role_remap_id:.`)

`__tag_role_remap_inline:` The mapping is meant to “degrade” tags, e.g. if used inside some complex object. The pdf<2.0 code maps the tag to the new role, the pdf 2.0 code only switch the NS.

```

638 \pdf_version_compare:NnTF < {2.0}
639 {
640   \cs_new_protected:Npn \__tag_role_remap_inline:
641   {
642     \prop_get:cVNT { g__tag_role_NS_latex-inline_prop } \l__tag_role_remap_tag_tl \l__tag_role_remap_NS_tl
643     {
644       \tl_set:Nc \l__tag_role_remap_tag_tl
645       {
646         \exp_last_unbraced:N \use_i:nn \l__tag_tmpa_tl
647       }
648       \tl_set:Nc \l__tag_role_remap_NS_tl
649       {
650         \exp_last_unbraced:N \use_ii:nn \l__tag_tmpa_tl
651       }

```



```

652     }
653     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
654     {
655         \msg_note:nne { tag } { role-remapping }{ \l__tag_role_remap_tag_tl }
656     }
657 }
658 }
659 {
660     \cs_new_protected:Npn \__tag_role_remap_inline:
661     {
662         \prop_get:cVNT { g__tag_role_NS_latex-inline_prop }{\l__tag_role_remap_tag_tl\l__tag_t
663         {
664             \tl_set:Nn\l__tag_role_remap_NS_tl {latex-inline}
665         }
666         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
667         {
668             \msg_note:nne { tag } { role-remapping }{ \l__tag_role_remap_tag_tl/latex-
inline }
669         }
670     }
671 }

```

(End of definition for __tag_role_remap_inline:.)

1.8 Key-val user interface

The user interface uses the key `add-new-tag`, which takes either a keyval list as argument, or a tag/role.

```

tag_(rolemap-key)
tag-namespace_(rolemap-key) 672 \keys_define:nn { __tag / tag-role }
role_(rolemap-key)          673 {
role-namespace_(rolemap-key) 674     ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
add-new-tag_(setup-key)     675     ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
                             676     ,role .tl_set:N = \l__tag_role_role_tmpa_tl
                             677     ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
                             678 }
                             679
680 \keys_define:nn { __tag / setup }
681 {
682     add-new-tag .code:n =
683     {
684         \keys_set_known:nnnN
685         {__tag/tag-role}
686         {
687             tag-namespace=user,
688             role-namespace=, %so that we can test for it.
689             #1
690             }{__tag/tag-role}\l_tmpa_tl
691         \tl_if_empty:NF \l_tmpa_tl
692         {
693             \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
694             \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
695             \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }

```

```

696     }
697     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
698     {
699         \prop_get:NVNTF
700         \g__tag_role_tags_NS_prop
701         \l__tag_role_role_tmpa_tl
702         \l__tag_role_role_namespace_tmpa_tl
703         {
704             \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
705             {
706                 \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
707             }
708         }
709         {
710             \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
711         }
712     }
713     \pdf_version_compare:NnTF < {2.0}
714     {
715         %TODO add check for emptyness?
716         \__tag_role_add_tag:VV
717         \l__tag_role_tag_tmpa_tl
718         \l__tag_role_role_tmpa_tl
719     }
720     {
721         \__tag_role_add_tag:VVVV
722         \l__tag_role_tag_tmpa_tl
723         \l__tag_role_tag_namespace_tmpa_tl
724         \l__tag_role_role_tmpa_tl
725         \l__tag_role_role_namespace_tmpa_tl
726     }
727 }
728 }
729 \end{package}

```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 150.)

Part X

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

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

`show-spaces_␣(setup-key)` This key works only with luatex and shows with small red bars where spaces have been inserted. This is only for debugging and is not completely reliable (and change affect other literals and tagging), so it should be used with care.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2023-12-18} {0.98r}
4 {part of tagpdf - code related to real space chars}
5 </header>
```

1 Code for interword spaces

The code is engine/backend dependant. Basically only pdftex and luatex support real space chars. Most of the code for luatex which uses attributes is in the lua code, here are only the keys.

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

```

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

```

(End of definition for `interwordspace` (setup-key) and `show-spaces` (setup-key). These functions are documented on page 171.)

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

```

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

```

```
75 }  
76 \end{package}  
  
(End of definition for \_tag\_fakespace:.)
```

Index

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

Symbols

\# 999, 1003
 \\ . 10, 23, 27, 28, 44, 45, 46, 53, 56, 58,
 64, 66, 76, 256, 257, 258, 395, 458, 466
 _ 431, 442

A

activate_ (setup-key) 34, 255
 activate-all_ (setup-key) 6, 236
 activate-mc_ (setup-key) 6, 236
 activate-socket_ (setup-key) 255
 activate-space_ (setup-key) 6, 236
 activate-struct_ (setup-key) 6, 236
 activate-tree_ (setup-key) 6, 236
 actualtext_ (mc-key) 66, 255, 453
 actualtext_ (struct-key) 97, 465
 add-new-tag_ (setup-key) 150, 672
 \AddToHook 13, 16, 57, 87, 273,
 348, 370, 395, 447, 465, 480, 509, 559
 AF_ (struct-key) 97, 580
 AFinline_ (struct-key) 97, 580
 AFinline-o_ (struct-key) 97, 580
 AFref_ (struct-key) 97, 580
 alt_ (mc-key) 66, 255, 453
 alt_ (struct-key) 96, 465
 artifact_ (mc-key) 66, 255, 453
 artifact-bool internal commands:
 __artifact-bool 121
 artifact-type internal commands:
 __artifact-type 121
 attr-unknown 19, 70
 attribute_ (struct-key) 98, 1146
 attribute-class_ (struct-key) .. 98, 1112

B

block internal commands:
 __block_debug_typeout:n 373, 385, 400
 __block_start_para_structure:n 372
 bool commands:
 \bool_gset_eq:NN ... 579, 594, 606, 624
 \bool_gset_false:N
 50, 51, 54, 238, 441, 580, 607
 \bool_gset_true:N
 47, 48, 49, 110, 177, 369
 \bool_if:NTF 9, 13, 18, 27,
 32, 36, 40, 69, 74, 79, 114, 192, 200,
 223, 223, 234, 238, 277, 278, 287,
 303, 327, 342, 373, 378, 390, 397,

404, 405, 428, 439, 449, 451, 467,
 473, 513, 574, 589, 601, 619, 806, 866
 \bool_if:nTF 6, 345
 \bool_if_exist_p:N 44
 \bool_lazy_all:nTF 101
 \bool_lazy_and:nnTF . 43, 135, 145,
 275, 283, 358, 426, 472, 500, 530, 593
 \bool_lazy_and_p:nn 8
 \bool_new:N 16, 20, 21,
 41, 42, 62, 105, 106, 107, 108, 109,
 111, 113, 115, 116, 229, 293, 294, 570
 \bool_set_false:N 178, 187, 188, 189,
 210, 211, 212, 239, 319, 546, 573, 600
 \bool_set_true:N ... 112, 114, 197,
 198, 199, 220, 221, 222, 230, 322, 545
 \box 367
 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 77, 177

C

\c 268, 269
 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 .. 136
 \c@g__tag_struct_abs_int
 ... 6, 17, 54, 77, 78, 81, 130, 147,
 150, 152, 229, 462, 477, 502, 514,
 528, 544, 552, 565, 575, 594, 597,
 602, 636, 638, 643, 655, 657, 662,
 715, 726, 727, 728, 729, 730, 731,
 734, 736, 742, 745, 762, 769, 787,
 796, 804, 832, 840, 845, 860, 861,
 863, 874, 972, 1035, 1139, 1142, 1190
 cctab commands:
 \c_document_cctab 73
 \chapter 160, 331, 343
 clist commands:
 \clist_const:Nn 102, 103
 \clist_if_empty:NTF 1151

G	
group commands:	
\group_begin:	66,
70, 175, 367, 441, 587, 677, 685, 725	
\group_end:	73,
73, 230, 419, 460, 605, 681, 689, 880	
H	
\hangindent	365
\hbox	356
hbox commands:	
\hbox_set:Nn	167, 168
hook commands:	
\hook_gput_code:nnn	7, 11,
30, 33, 43, 57, 137, 236, 258, 259,	
329, 340, 341, 344, 654, 667, 677, 690	
\hook_new:n	324
\hook_use:n	329
I	
\ignorespaces	34
int commands:	
\int_abs:n	142
\int_case:nnTF	82, 289
\int_compare:nNnTF	
.	22, 61, 68, 112, 118,
122, 135, 169, 170, 175, 200, 211,	
219, 246, 249, 274, 280, 344, 367,	
372, 374, 378, 381, 388, 401, 408,	
416, 423, 431, 436, 438, 442, 486,	
495, 630, 653, 666, 753, 819, 964, 1027	
\int_compare:nTF	
161, 314, 1132, 1134, 1136, 1160, 1186	
\int_compare_p:nNn	477
\int_decr:N	194, 217
\int_eval:n	134, 172, 291,
308, 349, 459, 467, 474, 479, 482,	
602, 643, 662, 727, 728, 729, 730,	
731, 734, 832, 860, 861, 863, 874, 1142	
\int_gincr:N	180,
239, 269, 308, 312, 316, 320, 326,	
330, 334, 338, 344, 351, 588, 715, 726	
\int_gset:Nn	45, 139, 287
\int_gzero:N	7, 295
\int_if_zero:nTF	
.	194, 195, 217, 218, 455, 463
\int_incr:N	56, 186, 209, 370
\int_new:N	41, 99,
104, 182, 263, 296, 297, 298, 299, 580	
\int_rand:n	61, 62, 64, 66, 68, 70, 71
\int_set:Nn	249, 252, 255, 256, 257
\int_step_inline:nn	54, 401, 407
\int_step_inline:nnn	25, 229
\int_step_inline:nnnn	
.	130, 155, 158, 175, 299, 305
\int_to_arabic:n	142, 144
\int_to_Hex:n	61, 62, 64, 66, 68, 70, 71
\int_use:N	
.	11, 15, 17, 24, 33, 46, 53, 64,
70, 73, 79, 80, 81, 83, 120, 122, 147,	
150, 152, 157, 159, 185, 202, 208,	
216, 225, 242, 251, 266, 274, 303,	
416, 431, 442, 462, 491, 492, 500,	
501, 502, 514, 528, 544, 552, 565,	
575, 591, 594, 597, 636, 638, 655,	
657, 736, 742, 745, 769, 787, 796,	
840, 845, 972, 1035, 1086, 1139, 1190	
\int_zero:N	53, 68, 358
intarray commands:	
\intarray_gset:Nnn	277, 355
\intarray_item:Nn	279, 282, 432
\intarray_new:Nn	269, 352
interwordspace _□ (setup-key)	171, 6
ior commands:	
\ior_close:N	304, 395
\ior_map_inline:Nn	300, 366
\ior_open:Nn	298, 361, 364
\g_tmpa_ior	
.	298, 300, 304, 361, 364, 366, 395
iow commands:	
\iow_newline:	201, 281, 456
\iow_now:Nn	74
\iow_term:n	181, 184, 190, 194, 194, 253
K	
keys commands:	
\keys_define:nn	7, 21, 30, 43, 99,
111, 121, 173, 216, 225, 237, 255,	
261, 413, 419, 454, 466, 547, 627,	
630, 672, 680, 692, 1105, 1112, 1146	
\keys_set:nn	10,
18, 96, 187, 266, 341, 345, 372, 493, 740	
\keys_set_known:nnnN	684
L	
label _□ (mc-key)	66, 255, 453
label _□ (struct-key)	96, 465
lang _□ (struct-key)	97, 465
legacy commands:	
\legacy_if:nTF	72, 374, 376
\llap	431
log _□ (setup-key)	6, 248
ltx. internal commands:	
ltx.__tag.func.alloctag	265
ltx.__tag.func.fakespace	439
ltx.__tag.func.fill_parent_tree_-	
line	803

paratag_□(setup-key) 413
 paratag_□(tool-key) 413
 paratagging_□(setup-key) 36, 413
 paratagging-show_□(setup-key) ... 36, 413
 parent_□(struct-key) 96, 465
 pdf commands:
 \pdf_activate_structure_destination:
 281
 \pdf_bdc:nn 232
 \pdf_bdc_shipout:nn 233
 \pdf_bmc:n 230
 \l_pdf_current_structure_
 destination_tl 279
 \pdf_emc: 231
 \pdf_name_from_unicode_e:n
 97, 107, 112,
 155, 164, 190, 251, 1102, 1120, 1156
 \pdf_object_if_exist:n 117
 \pdf_object_if_exist:nTF
 150, 184, 371, 634, 696
 \pdf_object_new:n 97,
 29, 34, 36, 135, 239, 286, 297, 733
 \pdf_object_ref:n 97,
 38, 56, 59, 96, 100, 117, 118, 129,
 152, 186, 192, 230, 294, 311, 375,
 434, 636, 698, 848, 945, 1008, 1049
 \pdf_object_ref_last:
 97, 67, 81, 87, 253, 1175
 \pdf_object_unnamed_write:nn ...
 63, 74, 83, 245, 1170
 \pdf_object_write:nnn 103,
 120, 232, 254, 287, 306, 313, 318, 389
 \pdf_pageobject_ref:n 198, 425
 \pdf_string_from_unicode:nnN ... 42
 \pdf_uncompress: 270
 \pdf_version_compare:NnTF
 20, 75, 84, 89,
 124, 147, 159, 210, 231, 237, 241,
 300, 317, 327, 359, 399, 473, 638, 713
 pdfannot commands:
 \pdfannot_dict_put:nnn
 119, 661, 684, 702, 707
 \pdfannot_link_ref_last: ... 671, 694
 pdfdict commands:
 \pdfdict_gput:nnn
 38, 45, 53, 187, 249, 310
 \pdfdict_if_empty:nTF 304
 \pdfdict_new:n 18, 35, 37
 \pdfdict_put:nnn ... 678, 679, 686, 687
 \pdfdict_use:n 256, 308, 315
 pdffakespace 36, 284
 pdffile commands:
 \pdffile_embed_stream:nnN .. 581, 589
 \pdffile_embed_stream:nnn 120
 \pdfglyptounicode 20
 \pdfinterwordspaceton 23, 24
 pdfmanagement commands:
 \pdfmanagement_add:nnn
 34, 35, 262, 264, 266, 346
 \pdfmanagement_if_active_p: .. 9, 10
 \pdfmanagement_remove:nn 268
 pdfmanagement internal commands:
 \l_pdfmanagement_delayed_
 shipout_bool 44, 45
 prg commands:
 \prg_do_nothing: 78, 85, 270,
 343, 344, 345, 346, 646, 647, 648, 649
 \prg_generate_conditional_
 variant:Nnn 117
 \prg_new_conditional:Nnn 66, 221
 \prg_new_conditional:Npnn
 95, 118, 133, 143, 337, 343, 354
 \prg_new_eq_conditional:NNn . 80, 228
 \prg_new_protected_conditional:Npnn
 616
 \prg_replicate:nn 141
 \prg_return_false: 76, 96, 113, 124,
 127, 140, 150, 225, 340, 352, 358, 631
 \prg_return_true: ... 77, 110, 123,
 137, 147, 224, 341, 351, 357, 616, 632
 \prg_set_conditional:Npnn 99
 \prg_set_protected_conditional:Npnn
 618
 \ProcessOptions 53
 prop commands:
 \prop_clear:N 157
 \prop_count:N 178
 \prop_get:NnN .. 140, 148, 179, 198,
 213, 242, 273, 384, 396, 398, 403,
 411, 412, 424, 425, 527, 528, 821, 1066
 \prop_get:NnNTF 92, 163,
 165, 178, 180, 183, 198, 217, 217,
 279, 373, 444, 480, 485, 490, 495,
 558, 578, 642, 662, 699, 781, 910, 992
 \prop_gput:Nnn 25,
 26, 30, 33, 34, 56, 91, 91, 92, 93,
 94, 94, 97, 99, 100, 101, 102, 103,
 113, 114, 115, 116, 121, 122, 123,
 124, 125, 151, 154, 163, 167, 201,
 205, 256, 259, 260, 288, 328, 347,
 375, 376, 376, 397, 403, 409, 415,
 417, 862, 873, 947, 1010, 1101, 1175
 \prop_gremove:Nn 135
 \prop_gset_eq:NN 134, 859
 \prop_if_exist:NnTF 294, 932, 989
 \prop_if_exist_p:N 474
 \prop_if_in:NnTF 69, 133, 155,
 163, 262, 351, 704, 1124, 1164, 1168

\prop_item:Nn	41, 73, 123, 167, 170, 220, 272, 356, 358, 361, 437, 445, 489, 1173, 1180
\prop_map_function:NN	235
\prop_map_inline:Nn	245, 302, 333, 345, 413
\prop_map_tokens:Nn	320
\prop_new:N	7, 8, 9, 10, 11, 11, 19, 24, 25, 32, 33, 64, 66, 95, 132, 165, 728, 1095, 1098
\prop_put:Nnn	122, 164, 478, 479, 551, 552
\prop_show:N	58, 91, 172, 856, 877, 1142, 1169
property commands:	
\property_gset:nnnn	128
\property_new:nnnn	127
\property_record:nn	134
\property_ref:nn	96, 130
\property_ref:nnn	129
\providecommand	62, 63, 64, 291, 511, 512
\ProvidesExplFile	3
\ProvidesExplPackage	3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 7, 26, 37, 1091
Q	
\quad	203, 204
quark commands:	
\q_no_value	487, 497, 570, 590
\quark_if_no_value:NTF	141, 149, 180, 199, 243, 274, 1073
\quark_if_no_value_p:N	427, 428, 501, 502, 531, 532, 594, 595
\q_stop	233, 266, 302
R	
raw_(mc-key)	66, 255, 453
ref_(struct-key)	97, 465
regex commands:	
\regex_replace_once:nnN	267
\RemoveFromHook	394
\renewcommand	545, 546
\RenewDocumentCommand	8
\RequirePackage	20, 54, 280, 283, 289, 292, 507
\rlap	442
role_(rolemap-key)	150, 672
role-missing	19, 72
role-namespace_(rolemap-key)	150, 672
role-parent-child	19, 75
role-remapping	19, 77
role-tag	19, 79
role-unknown	19, 72
role-unknown-tag	19, 72
root-AF_(setup-key)	98, 692
S	
\selectfont	6
seq commands:	
\seq_clear:N	279, 304
\seq_const_from_clist:Nn	20, 33
\seq_count:N	22, 25, 291, 383, 1132, 1134, 1136, 1160, 1186
\seq_get:NN	620
\seq_get:NTF	408, 440, 755, 899, 906
\seq_gpop:NN	892
\seq_gpop:NTF	105, 893
\seq_gpop_left:NN	266
\seq_gpush:Nn	12, 14, 88, 95, 762, 802
\seq_gput_left:Nn	271, 1128
\seq_gput_right:Nn	32, 144, 150, 168, 212, 232, 255, 324
\seq_gremove_duplicates:N	267
\seq_gset_eq:NN	155, 217, 286
\seq_if_empty:NTF	196, 377
\seq_item:Nn	112, 114, 121, 125, 132, 136, 169, 312, 347, 349, 356, 490, 491, 694, 695
\seq_log:N	171, 195, 219, 253, 411, 426
\seq_map_function:NN	244
\seq_map_indexed_inline:Nn	374, 387
\seq_map_inline:Nn	280, 341, 1122, 1162
\seq_new:N	11, 13, 14, 15, 16, 17, 17, 18, 18, 18, 96, 97, 133, 166, 731, 1096
\seq_pop_left:NN	383, 385, 386
\seq_put_right:Nn	281
\seq_remove_all:Nn	284
\seq_set_eq:NN	203, 204
\seq_set_from_clist:NN	1117, 1153
\seq_set_from_clist:Nn	83, 86, 192, 212, 371, 382
\seq_set_map:NNn	268
\seq_set_map_e:NNn	1118, 1154
\seq_set_split:Nnn	124, 489, 693
\seq_show:N	51, 171, 186, 187, 220, 282, 283, 285, 334, 805, 857, 878, 888
\seq_use:Nn	45, 106, 107, 201, 203, 204, 279, 326, 1133
\l_tmpa_seq	304, 324, 334, 693, 694, 695
\setbox	355
shipout commands:	
\g_shipout_readonly_int	79, 157, 216, 349
show-kids	19, 50
show-spaces_(setup-key)	171, 6
show-struct	19, 50
\ShowTagging	16, 35, 93

skip commands:

- \skip_horizontal:n 72
- \c_zero_skip 72
- stash_(mc-key) 66, 121
- stash_(struct-key) 96, 465
- \stepcounter 437

str commands:

- \str_case:nnTF 59, 774
- \str_const:Nn 59
- \str_if_empty:nTF 599
- \str_if_eq:nnTF ... 123, 356, 442, 529
- \str_if_eq_p:nn 284, 347, 349
- \str_new:N 94
- \str_set_convert:Nnnn .. 125, 278, 299, 467, 480, 496, 508, 522, 538, 569
- \str_use:N 289, 312
- \string 20, 21, 22, 531
- struct-faulty-nesting 19, 32
- struct-label-unknown 19, 38
- struct-missing-tag 19, 35
- struct-no-objnum 19, 24
- struct-orphan 19, 25
- struct-show-closing 19, 40
- struct-stack_(show-key) 35, 216
- struct-unknown 19, 22
- struct-used-twice 19, 36
- \SuspendTagging 37

sys commands:

- \c_sys_backend_str 59
- \c_sys_engine_str 10, 12
- \sys_if_engine luatex:TF 41, 49, 66, 86, 103, 108, 116, 272, 284
- \sys_if_engine pdftex:TF 16
- \sys_if_output_pdf:TF 11, 18
- sys-no-interwordspace 19, 86

T

taborder_(setup-key) 6, 260

tag_(mc-key) 66, 255, 453

tag_(rolemap-key) 150, 672

tag_(struct-key) 96, 465

tag commands:

- \tag_check_child:nn ... 150, 616, 618
- \tag_check_child:nnTF 150, 616
- \tag_get:n 16, 67, 95, 96, 111, 88, 91, 93, 93, 395
- \tag_if_active: 95, 99
- \tag_if_active:TF 16, 18, 94, 482
- \tag_if_active_p: 16, 94, 359
- \tag_if_box_tagged:N 16, 118
- \tag_if_box_tagged:NTF 16, 117
- \tag_if_box_tagged_p:N 16, 117
- \tag_mc_artifact_group_begin:n .. 65, 59, 59, 62
- \tag_mc_artifact_group_end: 65, 59, 60, 70
- \tag_mc_begin:n ... 10, 65, 25, 65, 113, 171, 171, 351, 351, 355, 361, 392, 430, 441, 462, 582, 610, 660, 683
- \tag_mc_begin_pop:n 65, 75, 79, 80, 101, 591, 621, 674, 697
- \tag_mc_end: 65, 31, 74, 92, 233, 233, 351, 352, 401, 428, 432, 434, 443, 470, 588, 617, 672, 695
- \tag_mc_end_push: 65, 64, 79, 79, 82, 576, 603, 658, 681
- \tag_mc_if_in: 80, 228
- \tag_mc_if_in:TF 65, 42, 66, 221
- \tag_mc_if_in_p: 65, 66, 221
- \tag_mc_reset_box:N 66, 78, 78, 245, 245
- \tag_mc_use:n 65, 35, 35, 36, 37
- \l_tag_para_attr_class_tl 389
- \tag_socket_use:n .. 37, 38, 62, 66, 67
- \tag_socket_use:nn . 37, 38, 63, 66, 72
- \tag_start: 6, 181, 192, 205, 230
- \tag_start:n 6, 72, 181, 215, 234, 366, 587, 616
- \tag_stop: ... 6, 46, 181, 183, 204, 229
- \tag_stop:n 6, 67, 181, 206, 233, 364, 583, 611
- \tag_struct_begin:n 95, 48, 381, 386, 454, 460, 609, 659, 682, 715, 715, 719, 720
- \tag_struct_end: 95, 26, 53, 403, 407, 472, 476, 618, 673, 696, 715, 716, 884, 885, 923
- \tag_struct_end:n 95, 717, 920
- \tag_struct_gput:nnn 96, 1052, 1052, 1060
- \tag_struct_insert_annot:nn ... 95, 125, 671, 694, 1076, 1076, 1085
- \tag_struct_object_ref:n 95, 1046, 1047, 1051
- \tag_struct_parent_int: 95, 125, 664, 671, 687, 694, 1076, 1086
- \tag_struct_use:n 95, 96, 58, 926, 926, 928
- \tag_struct_use_num:n 95, 983, 983, 985
- \tag_tool:n 34, 13, 13, 14, 16, 20

tag internal commands:

- __tag_activate_mark_space 503
- \g__tag_active_mc_bool 40, 104, 105, 135, 240
- \l__tag_active_mc_bool 107, 111, 135, 188, 198, 211, 221
- \l__tag_active_socket_bool .. 69, 74, 79, 111, 189, 199, 212, 222, 263

\g__tag_active_space_bool	13, 48, 54, 105 , 239	__tag_check_info_closing_-	
\g__tag_active_struct_bool	103, 105 , 145, 242, 277, 405	struct:n	168 , 168, 176, 895
\l__tag_active_struct_bool	106, 111 , 145, 187, 197, 210, 220	__tag_check_init_mc_used:	267, 267, 270, 276
\g__tag_active_struct_dest_bool	105 , 246, 276	__tag_check_mc_if_nested:	176, 229 , 229, 368
\g__tag_active_tree_bool	9, 32, 105 , 105, 241, 327, 342	__tag_check_mc_if_open:	229 , 237, 237, 440
__tag_add_missing_mcs:Nn	78, 163 , 163, 215	__tag_check_mc_in_galley:TF ..	337
__tag_add_missing_mcs_to_-		__tag_check_mc_in_galley_p: ..	337
stream:Nn	65,	__tag_check_mc_pushed_popped:n	89, 96, 109, 112, 117, 244 , 244
65, 185 , 185, 518, 522, 527, 534, 536		__tag_check_mc_tag:N	189, 256 , 256, 380
\g__tag_attr_class_used_seq	267, 268, 1094 , 1128	__tag_check_mc_used:n	143, 272 , 272, 324
\g__tag_attr_entries_prop	273, 1094 , 1101, 1124, 1164, 1169, 1173	\g__tag_check_mc_used_intarray ..	267 , 277, 279, 282
__tag_attr_new_entry:nn	597, 1099 , 1099, 1109	__tag_check_no_open_struct: ..	177, 177, 897, 904
\g__tag_attr_objref_prop	1094 , 1168, 1175, 1180	__tag_check_para_begin_show:nn ..	391, 425, 461
\l__tag_attr_value_tl	1094 ,	__tag_check_para_end_show:nn ..	402, 436, 471
1158, 1177, 1182, 1184, 1188, 1192		__tag_check_parent_child:nnN ..	519, 525, 613
__tag_backend_create_bdc_node ..	389	__tag_check_parent_child:nnnnN ..	473
__tag_backend_create_bmc_node ..	360	__tag_check_parent_child:nnnnN ..	206, 396, 475,
__tag_backend_create_emc_node ..	331	521, 534, 549, 614, 625, 813, 958, 1021	
__tag_check_add_tag_role:nn ..	132, 191 , 191	__tag_check_show_MCID_by_page: ..	291 , 291
__tag_check_add_tag_role:nnn ..	174, 210	__tag_check_struct_used:n	181 , 181, 935
__tag_check_if_active_mc:	133	__tag_check_structure_has_tag:n	153 , 153, 745
__tag_check_if_active_mc:TF ..	84, 103,	__tag_check_structure_tag:N ..	161 , 161, 492
132, 173, 187, 235, 357, 363, 430, 436		__tag_check_typeout_v:n	88, 88, 106, 107, 110, 145,
__tag_check_if_active_struct: ..	143	153, 160, 198, 207, 253, 521, 526, 531	
__tag_check_if_active_struct:TF		__tag_debug_mc_begin_ignore:n ..	372, 423
39, 132 ,		__tag_debug_mc_begin_insert:n ..	365, 365
722, 723, 889, 890, 922, 930, 987, 1079		__tag_debug_mc_end_ignore: ..	386, 448
__tag_check_if_mc_in_galley: ..	337	__tag_debug_mc_end_insert: ..	379, 438
__tag_check_if_mc_in_galley:TF ..	179, 200	__tag_debug_struct_begin_-	
__tag_check_if_mc_tmb_missing: ..	343	ignore:n	414, 882
__tag_check_if_mc_tmb_missing:TF		__tag_debug_struct_begin_-	
108, 188, 205, 343		insert:n	406, 879
__tag_check_if_mc_tmb_missing_-		__tag_debug_struct_end_check:n ..	436, 922
p:	343		
__tag_check_if_mc_tme_missing: ..	354		
__tag_check_if_mc_tme_missing:TF			
151, 192, 209, 354			
__tag_check_if_mc_tme_missing_-			
p:	354		

__tag_debug_struct_end_ignore: .	__tag_hook_kernel_after_foot: ..
..... 429, 917 557, 566, 635, 642, 649
__tag_debug_struct_end_insert: .	__tag_hook_kernel_after_head: ..
..... 421, 915 555, 564, 634, 641, 648
\g__tag_delayed_shipout_bool ...	__tag_hook_kernel_before_foot: .
..... 42, 47, 51, 234 556, 565, 633, 640, 647
__tag_exclude_headfoot_begin: ..	__tag_hook_kernel_before_head: .
..... 571, 632, 633 554, 563, 632, 639, 646
__tag_exclude_headfoot_end: ...	\g__tag_in_mc_bool
..... 585, 634, 635 16, 18, 177, 223, 238,
__tag_exclude_struct_headfoot_-	369, 441, 579, 580, 594, 606, 607, 624
begin:n	__tag_insert_bdc_node
598, 639, 640	389
__tag_exclude_struct_headfoot_-	__tag_insert_bmc_node
end:	360
614, 641, 642	__tag_insert_emc_node
__tag_fakespace	331
439	__tag_lastpagelabel:
__tag_fakespace:	69, 70, 88
66, 68, 288	__tag_log
__tag_finish_structure:	177
.....	\l__tag_loglevel_int
13, 16, 324, 325 104, 135, 169, 170, 175,
__tag_get_data_mc_counter: ...	200, 219, 247, 249, 250, 252, 255,
9, 9	256, 257, 274, 367, 374, 381, 388,
__tag_get_data_mc_tag:	408, 416, 423, 431, 438, 442, 653, 666
.....	__tag_mark_spaces
254, 254, 349, 349	444
__tag_get_data_struct_counter: .	__tag_mc_artifact_begin_marks:n
..... 19, 41, 77, 377
459, 460	\l__tag_mc_artifact_bool
__tag_get_data_struct_id: 20, 124, 178, 192, 239, 373
448, 448	\l__tag_mc_artifact_type_tl
__tag_get_data_struct_num: 453, 454 19, 128, 132, 136,
__tag_get_data_struct_tag: 440, 440	140, 144, 148, 152, 156, 347, 375, 377
__tag_get_mathsubtype	__tag_mc_bdc:nn 229, 232, 264, 306, 339
255	__tag_mc_bdc_mcid:n .. 119, 234, 311
__tag_get_mc_abs_cnt:	__tag_mc_bdc_mcid:nn
..... 14, 15, 19, 20, 234, 237, 267, 313, 318
100, 105, 135, 146, 185, 227, 233,	__tag_mc_bdc_shipout:nn ... 233, 245
241, 260, 263, 271, 289, 310, 324, 334	__tag_mc_begin_marks:nn
__tag_get_mc_cnt_type_tag 249 19, 19, 40, 76, 384
__tag_get_num_from	__tag_mc_bmc:n
274	229, 230, 335
\l__tag_get_parent_tmpa_tl	__tag_mc_bmc_artifact: 333, 333, 346
..... 92, 204, 207, 220, 394,	__tag_mc_bmc_artifact:n 333, 337, 347
397, 410, 623, 626, 811, 814, 828, 870	\l__tag_mc_botmarks_seq
\l__tag_get_parent_tmpa_tl\l__- 78, 17, 86, 107,
tag_get_parent_tmpb_tl\l__-	157, 187, 204, 204, 212, 217, 339, 356
tag_tmpa_str	__tag_mc_disable_marks: 74, 74
89	__tag_mc_emc: 154, 229, 231, 443
\l__tag_get_parent_tmpb_tl	__tag_mc_end_marks: . 19, 59, 78, 444
..... 93, 205, 208, 220,	\l__tag_mc_firstmarks_seq
395, 398, 410, 624, 627, 812, 815, 828 77, 17, 83, 106, 186, 192,
__tag_get_tag_from	195, 196, 203, 203, 204, 339, 347, 349
293	\g__tag_mc_footnote_marks_seq ... 14
\l__tag_get_tmpe_tl 89, 165,	__tag_mc_get_marks: 80, 80, 178, 199
170, 181, 183, 184, 784, 790, 1069, 1073	__tag_mc_handle_artifact:N
__tag_gincr_para_begin_int: 115, 333, 341, 375
..... 306, 310, 328, 344, 353, 384, 459	__tag_mc_handle_mc_label:n
__tag_gincr_para_end_int: 26, 26, 197, 388
..... 306, 318, 336, 346, 399, 469	
__tag_gincr_para_main_begin_-	
int: ... 306, 306, 324, 343, 380, 453	
__tag_gincr_para_main_end_int: .	
..... 306, 314, 332, 345, 406, 475	

__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, 231, 239	
__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, 268
__tag_mc_insert_mcid_single_kids:n	129, 134, 269
\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, 167, 328
\l_tag_mc_ref_abbrevpage_tl	11, 270, 282, 290, 298
__tag_mc_set_label_used:n	30, 30, 50	
\g_tag_mc_stack_seq	18, 88, 95, 105, 253
__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_byabbrevpage_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, 223, 277, 278, 287, 303, 360, 513, 574, 589, 601, 619
__tag_new_output_prop_handler:n	67, 77, 101, 729
__tag_pairs_prop	194
\g_tag_para_begin_int	292, 312, 330, 431, 495, 500
\l_tag_para_bool	292, 397, 415, 449, 467, 545, 546, 549, 573, 600
\g_tag_para_end_int	292, 320, 338, 442, 495, 501
\l_tag_para_flattened_bool	...	292, 378, 404, 423, 451, 473, 550
\g_tag_para_main_begin_int	...	292, 308, 326, 486, 491
\g_tag_para_main_end_int	292, 316, 334, 486, 492
\l_tag_para_main_tag_tl	292, 381, 422, 456
\l_tag_para_show_bool	292, 416, 428, 439
\l_tag_para_tag_default_tl	...	292
\l_tag_para_tag_tl	292, 352, 388, 417, 421, 460
\l_tag_parent_child_check_tl	...	210, 211, 400, 401, 419, 629, 630, 818, 819, 963, 964, 1026, 1027
__tag_parenttree_add_objr:nn	...	144, 144, 429
\l_tag_parenttree_content_tl	...	151, 170, 182, 202, 210, 231, 234
\g_tag_parenttree_objr_tl	143, 146, 231
__tag_pdf_name_e:n	97, 97
__tag_pdf_object_ref	419
__tag_prop_gput:Nnn	9, 23, 84, 89, 93, 97, 114, 165, 167, 174, 288, 291, 296, 941
__tag_prop_item:Nn	.. 9, 43, 165, 170	
__tag_prop_new:N	9, 9, 10, 17, 100, 165, 165, 176, 262, 727
__tag_prop_show:N	9, 56, 165, 172, 179	
__tag_property_gset:nnnn	127, 128, 265
\c_tag_property_mc_clist	102, 244, 305
__tag_property_new:nnnn	127, 127, 145, 148, 155, 158, 162
__tag_property_record:nn	28, 131, 140, 240, 301, 416, 749
__tag_property_ref:nn	.. 130, 139, 562	
__tag_property_ref:nnn	41, 127, 129, 138, 143, 162, 166, 184, 198, 199, 272, 316, 327, 425, 933, 939, 942, 948, 955
__tag_property_ref_lastpage:nn	.. 46, 141, 141, 141, 155, 158, 295, 309	
\c_tag_property_struct_clist	...	102, 751

<code>g__tag_role/RoleMap_dict</code>	18	<code>\l__tag_role_tag_namespace_tmprb-</code>	
<code>__tag_role_add_tag:nn</code>		<code>tl</code>	14 , 528 , 529 , 532 , 536
.....	130 , 130 , 158 , 254 , 335 , 716	<code>\l__tag_role_tag_namespace_tmprb-</code>	
<code>__tag_role_add_tag:nnnn</code>		<code>tluuuuuu%</code>	12
.....	172 , 172 , 209 , 286 , 721	<code>\l__tag_role_tag_tmprb_tl</code>	
<code>__tag_role_alloctag:nnn</code>	84 ,	12 , 674 , 694 , 717 , 722
88 , 98 , 110 , 120 , 129 , 145 , 184 , 251 , 282		<code>\g__tag_role_tags_class_prop</code> ...	
<code>\l__tag_role_debug_prop</code>	151 , 8 , 93 , 102 , 115 , 124 , 140 , 242
.....	151 , 11 , 478 , 479 , 551 , 552	<code>\g__tag_role_tags_NS_prop</code>	
<code>__tag_role_get:nnNN</code>	151 , 7 , 91 , 100 , 113 , 122 , 133 , 163 ,
...	159 , 161 , 169 , 210 , 212 , 227 , 763	...	198 , 262 , 347 , 489 , 527 , 528 , 700 , 910
<code>__tag_role_get_parent_child-</code>		<code>\l__tag_role_tmprb_seq</code>	12
<code>rule:nnnN</code>	164 , 419 , 420 , 472 , 504 , 597	<code>\l__tag_role_update_bool</code>	
<code>\g__tag_role_index_prop</code> .	151 , 10 ,	229 , 230 , 238 , 319 , 322
376 , 384 , 396 , 397 , 398 , 403 , 409 ,		<code>\c__tag_role_userNS_id_str</code>	
411 , 412 , 415 , 417 , 424 , 425 , 480 , 490		152 , 59 , 83
<code>\g__tag_role_NS<ns>_class_prop</code>	151	<code>\g__tag_root_default_tl</code>	255
<code>\g__tag_role_NS<ns>_prop</code>	151	<code>\g__tag_saved_in_mc_bool</code>	
<code>\g__tag_role_NS_mathml_prop</code> ...	413	570 , 579 , 594 , 606 , 624
<code>__tag_role_NS_new:nnn</code>	153 ,	<code>__tag_seq_gput_right:Nn</code>	9 ,
20 , 22 , 30 , 73 , 74 , 77 , 79 , 80 , 81 , 83		...	30 , 165 , 168 , 175 , 207 , 217 , 227 , 250
<code>\g__tag_role_NS_prop</code>		<code>__tag_seq_item:Nn</code> ...	9 , 38 , 165 , 169
...	151 , 9 , 26 , 56 , 165 , 302 , 320 , 704	<code>__tag_seq_new:N</code>	
<code>\g__tag_role_parent_child-</code>		...	9 , 9 , 16 , 102 , 165 , 166 , 177 , 730
<code>intarray</code>	352 , 355 , 433	<code>__tag_seq_show:N</code> .	9 , 49 , 165 , 171 , 178
<code>__tag_role_read_namespace:n</code> ...		<code>__tag_show_spacemark</code>	425
.....	311 , 311 ,	<code>\l__tag_showspaces_bool</code> ...	14 , 26 , 35
315 , 316 , 318 , 320 , 321 , 323 , 325 , 326		<code>__tag_space_chars_shipout</code>	534
<code>__tag_role_read_namespace:nn</code> ...		<code>__tag_start_para_ints:</code>	
.....	292 , 292 , 313 , 324	200 , 223 , 322 , 322
<code>__tag_role_read_namespace-</code>		<code>__tag_stop_para_ints:</code>	
<code>line:nw</code>	229 , 233 , 266 , 302	190 , 213 , 322 , 341
<code>__tag_role_remap:</code>		<code>__tag_store_parent_child-</code>	
.....	636 , 636 , 637 , 836 , 968 , 1031	<code>rule:nnn</code>	353 , 353 , 390
<code>__tag_role_remap_id:</code>	637 , 637	<code>g__tag_struct_0_prop</code>	99
<code>__tag_role_remap_inline:</code>		<code>__tag_struct_add_AF:nn</code>	
.....	638 , 640 , 660	593 , 610 , 629 , 636 , 655 , 698
<code>\l__tag_role_remap_NS_tl</code> ...	634 ,	<code>__tag_struct_add_inline_AF:nn</code> ..	
648 , 664 , 835 , 838 , 967 , 970 , 1030 , 1033		582 , 609 , 669 , 673 , 680 , 688
<code>\l__tag_role_remap_tag_tl</code>		<code>\g__tag_struct_AFobj_int</code>	580 , 588 , 591
.....	634 , 642 , 644 , 655 ,	<code>\g__tag_struct_cont_mc_prop</code>	
662 , 668 , 834 , 837 , 966 , 969 , 1029 , 1032		10 , 91 , 92 , 94 , 97 , 220
<code>\l__tag_role_role_namespace-</code>		<code>\g__tag_struct_dest_num_prop</code> ...	63
<code>tmprb_tl</code>	12 ,	<code>\l__tag_struct_elem_stash_bool</code> ..	
677 , 697 , 702 , 704 , 706 , 710 , 725		62 , 469 , 807 , 866
<code>\l__tag_role_role_tmprb_tl</code>		<code>__tag_struct_exchange_kid-</code>	
.....	12 , 676 , 695 , 701 , 718 , 724	<code>command:N</code>	264 , 264 , 274 , 305
<code>\g__tag_role_rolemap_prop</code> ..	151 ,	<code>__tag_struct_fill_kid_key:n</code> ...	
18 , 148 , 151 , 154 , 163 , 245 , 485 , 495		101 , 118 , 275 , 275 , 386
<code>\c__tag_role_rules_num_prop</code>	353 , 444	<code>__tag_struct_format_Ref:n</code>	
<code>\c__tag_role_rules_prop</code>	353 , 356 , 437	367 , 367 , 368
<code>\l__tag_role_tag_namespace_tmprb-</code>		<code>__tag_struct_get_dict_content:nN</code>	
<code>tl</code>	12 , 527 , 531 , 535 , 675 , 723	102 , 119 , 338 , 338 , 387


```

\__tag_struct_get_id:n .....
.... 59, 64, 77, 78, 136, 137, 394, 450
\__tag_struct_get_parentrole:nnN
..... 175,
175, 191, 202, 392, 621, 809, 954, 1017
\__tag_struct_gput_data_ref:nn ..
..... 564, 1062, 1063, 1075
\__tag_struct_insert_annot:mn ....
..... 402, 402, 1081
\l__tag_struct_key_label_tl .....
..... 61, 468, 747, 750
\__tag_struct_kid_mc_gput_-
right:nn ... 192, 204, 205, 223, 325
\__tag_struct_kid_OBJR_gput_-
right:nnn .. 240, 240, 243, 263, 417
\__tag_struct_kid_struct_gput_-
right:nn .....
... 224, 224, 225, 239, 853, 937, 1000
g__tag_struct_kids_0_seq ..... 99
\__tag_struct_mcid_dict:n .....
..... 94, 97, 192, 210
\g__tag_struct_objR_seq ..... 8
\__tag_struct_output_prop_aux:nn
..... 67, 67, 81
\__tag_struct_prop_gput:nnn ....
..... 85, 86, 87, 93, 104,
109, 114, 119, 126, 152, 161, 167,
308, 321, 501, 513, 527, 543, 551,
574, 596, 637, 656, 699, 735, 768,
786, 795, 844, 1004, 1070, 1138, 1189
\g__tag_struct_ref_by_dest_prop . 66
\g__tag_struct_roletag_NS_tl ... 57
\l__tag_struct_roletag_NS_tl ...
..... 60, 767, 772, 799
\l__tag_struct_roletag_tl .....
..... 57, 766, 772, 774, 799, 803
\__tag_struct_set_tag_info:nnn ..
147, 149, 159, 174, 741, 839, 971, 1034
\g__tag_struct_stack_current_tl .
..... 15, 25, 34, 65, 71, 96, 146,
152, 160, 166, 203, 214, 224, 280,
326, 330, 393, 404, 413, 445, 450,
456, 804, 851, 855, 856, 877, 895,
901, 938, 945, 951, 1001, 1008, 1014
\l__tag_struct_stack_parent_-
tmpa_tl ..... 15, 410, 419,
434, 479, 739, 753, 757, 782, 810,
822, 831, 848, 852, 854, 857, 869, 878
\g__tag_struct_stack_seq 11, 22, 25,
409, 620, 756, 762, 805, 888, 893, 899
\c__tag_struct_StructElem_-
entries_seq ..... 20
\c__tag_struct_StructTreeRoot_-
entries_seq ..... 20
\g__tag_struct_tag_NS_tl .....
..... 57, 491, 744, 765, 817,
829, 835, 838, 842, 876, 912, 960,
967, 970, 974, 1023, 1030, 1033, 1037
\g__tag_struct_tag_stack_seq ...
..... 13, 45,
219, 220, 411, 426, 440, 802, 892, 906
\g__tag_struct_tag_tl .....
. 57, 181, 182, 185, 370, 371, 490,
492, 743, 764, 803, 816, 829, 834,
837, 841, 908, 910, 952, 959, 966,
969, 973, 1015, 1022, 1029, 1032, 1036
\__tag_struct_write_obj:n .....
..... 132, 369, 369
\l__tag_tag_stop_int 181, 185, 186,
194, 195, 202, 208, 209, 217, 218, 225
\g__tag_tagunmarked_bool ... 116, 258
\l__tag_tmpa_box .....
..... 89, 167, 173, 174, 178, 189, 190
\l__tag_tmpa_clist .....
..... 89, 1116, 1117, 1150, 1151, 1153
\l__tag_tmpa_int ..... 53,
56, 61, 64, 68, 77, 89, 358, 370, 372, 442
\l__tag_tmpa_prop 89, 157, 165, 178, 180
\l__tag_tmpa_seq .... 89, 268, 279,
280, 281, 283, 284, 285, 286, 371,
374, 382, 383, 385, 386, 387, 489,
490, 491, 1118, 1122, 1132, 1133,
1134, 1136, 1154, 1160, 1162, 1186
\l__tag_tmpa_str ..... 42,
43, 48, 94, 279, 284, 289, 300, 305,
312, 468, 473, 481, 486, 497, 504,
509, 516, 523, 530, 539, 546, 570, 577
\l__tag_tmpa_tl ... 41, 42, 49, 51,
57, 65, 69, 72, 79, 84, 89, 91, 92, 94,
102, 105, 107, 107, 112, 113, 114,
115, 119, 124, 140, 141, 143, 145,
148, 149, 154, 179, 180, 180, 181,
182, 184, 184, 186, 186, 198, 199,
205, 216, 217, 219, 220, 224, 242,
243, 245, 249, 251, 266, 266, 270,
271, 273, 274, 276, 277, 280, 282,
284, 290, 293, 301, 383, 384, 385,
386, 387, 393, 396, 397, 398, 403,
406, 409, 411, 413, 415, 424, 427,
434, 440, 444, 444, 446, 448, 455,
480, 482, 485, 487, 501, 505, 555,
558, 561, 561, 563, 564, 565, 566,
570, 592, 594, 595, 598, 620, 622,
642, 646, 650, 662, 824, 831, 892,
893, 899, 901, 906, 909, 910, 912,
956, 961, 995, 1019, 1024, 1130, 1141
\l__tag_tmpb_box .....
..... 89, 168, 175, 176, 180, 182

```

<code>\l__tag_tmpb_seq</code>	<code>\tagstructbegin</code>
..... 89 , 1117 , 1118 , 1153 , 1154 35 , 150 , 151 , 45 , 258 , 352 , 354
<code>\l__tag_tmpb_tl</code>	<code>\tagstructend</code>
81 , 83 , 89 , 373 , 384 , 390 , 412 , 417 ,	35 , 45 , 259 , 367
425 , 428 , 434 , 448 , 490 , 492 , 495 ,	<code>tagstructobj</code>
497 , 502 , 505 , 575 , 581 , 583 , 584 ,	6 , 145
586 , 590 , 595 , 598 , 957 , 962 , 1020 , 1025	<code>\tagstructuse</code>
<code>__tag_tree_fill_parenttree:</code> ...	35 , 45
..... 152 , 153 , 228	<code>\tagtool</code>
<code>__tag_tree_final_checks:</code> 20 , 20 , 330	34 , 13
<code>\g__tag_tree_id_pad_int</code> .. 41 , 45 , 142	<code>tagunmarked_␣(setup-key)</code>
<code>__tag_tree_lua_fill_parenttree:</code>	6 , 258
..... 208 , 208 , 225	TeX and L ^A T _E X 2 _ε commands:
<code>__tag_tree_parenttree_rerun-</code>	<code>\@M</code>
<code>msg:</code>	164
152 , 195 , 230	<code>\@auxout</code>
<code>__tag_tree_write_classmap:</code>	74
..... 264 , 264 , 334	<code>\@bsphack</code>
<code>__tag_tree_write_idtree:</code> ... 49 , 332	133
<code>__tag_tree_write_namespaces:</code> ...	<code>\@cclv</code>
..... 298 , 298 , 335	522
<code>__tag_tree_write_parenttree:</code> ...	<code>\@esphack</code>
..... 221 , 221 , 331	135
<code>__tag_tree_write_rolemap:</code>	<code>\@gobble</code>
..... 241 , 243 , 261 , 333	31 , 55
<code>__tag_tree_write_structelements:</code>	<code>\@ifpackageloaded</code>
..... 128 , 128 , 336	28 , 505
<code>__tag_tree_write_structtreeroot:</code>	<code>\@kernel@after@foot</code>
..... 89 , 91 , 112 , 337	566
<code>__tag_whatsits:</code> 35 , 61 , 62 , 65 , 351 , 352	<code>\@kernel@after@head</code>
<code>tag-namespace_␣(rolemap-key)</code>	564
672	<code>\@kernel@before@cclv</code>
<code>tag/struct/0</code> internal commands:	512 , 519
<code>__tag/struct/0</code>	565
29	<code>\@kernel@before@footins</code>
<code>tag/tree/namespaces</code> internal commands:	515 , 517
<code>__tag/tree/namespaces</code>	561 , 563
297	<code>\@kernel@tag@hangfrom</code>
<code>tag/tree/parenttree</code> internal commands:	350
<code>__tag/tree/parenttree</code>	<code>\@kernel@tagsupport@makecol</code> 511 , 524
135	<code>\@makecol</code>
<code>tag/tree/rolemap</code> internal commands:	521 , 526
<code>__tag/tree/rolemap</code>	<code>\@maxdepth</code>
237	177
<code>tagabspage</code>	<code>\@mult@ptagging@hook</code>
6 , 145	529
<code>tagmcabs</code>	<code>\@outputbox</code>
6 , 145	527
<code>\tagmcbegin</code>	31 , 55
34 , 151 , 22 , 361 , 367	<code>\@secondoftwo</code>
<code>\tagmcend</code>	355 , 365 , 367
34 , 22 , 367	<code>\@tempboxa</code>
<code>tagmcid</code>	521 , 526
6 , 145	<code>\c@page</code>
<code>\tagmcifin</code>	534
34	<code>\count@</code>
<code>\tagmcifinTF</code>	532
34 , 39	<code>\mult@firstbox</code>
<code>\tagmcuse</code>	536
34 , 22	<code>\new@label@record</code>
<code>\tagpdfparaOff</code>	76
36 , 542	<code>\on@line</code>
<code>\tagpdfparaOn</code>	375 , 385 , 400
36 , 542	<code>\page@sofar</code>
<code>\tagpdfsetup</code>	531
34 , 98 , 150 , 6	<code>\process@cols</code>
<code>\tagpdfsuppressmarks</code>	532
36 , 552	tex commands:
<code>\tagstart</code>	<code>\tex_botmarks:D</code>
6 , 205 , 232	87
<code>\tagstop</code>	<code>\tex_firstmarks:D</code>
6 , 204 , 231	84
<code>tagstruct</code>	<code>\tex_kern:D</code>
6 , 145	180
	<code>\tex_marks:D</code>
	21 , 30 , 43 , 50 , 61 , 67
	<code>\tex_special:D</code>
	65
	<code>\tex_splitbotmarks:D</code>
	213
	<code>\tex_splitfirstmarks:D</code>
	193
	<code>texsource</code>
	97
	<code>\the</code>
	521 , 526
	<code>\tiny</code>
	431 , 442
	<code>title_␣(struct-key)</code>
	96 , 465
	<code>title-o_␣(struct-key)</code>
	96 , 465
	tl commands:
	<code>\c_empty_tl</code>
	335
	<code>\c_space_tl</code>
	67 , 73 , 148 , 172 ,
	173 , 191 , 191 , 195 , 197 , 199 , 234 ,

