

tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer[†]

Released 2024-06-20

Contents

1	Initialization and test if pdfmanagement is active.	7
2	base package	7
3	Package options	8
4	Packages	8
	4.1 Indexed objects	8
	4.2 a LastPage label	9
5	Variables	9
6	Variants of l3 commands	11
7	Label and Reference commands	11
8	Setup label attributes	12
9	Commands to fill seq and prop	12
10	General tagging commands	13
11	Keys for tagpdfsetup	14
12	loading of engine/more dependent code	16
I	The tagpdf-checks module	
	Messages and check code	
	Part of the tagpdf package	17
1	Commands	17

*This file describes v0.99c, last revised 2024-06-20.

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

2	Description of log messages	17
2.1	\ShowTagging command	17
2.2	Messages in checks and commands	18
2.3	Messages from the ptagging code	18
2.4	Warning messages from the lua-code	18
2.5	Info messages from the lua-code	18
2.6	Debug mode messages and code	19
2.7	Messages	19
3	Messages	21
3.1	Messages related to mc-chunks	21
3.2	Messages related to structures	22
3.3	Attributes	24
3.4	Roles	24
3.5	Miscellaneous	24
4	Retrieving data	25
5	User conditionals	25
6	Internal checks	26
6.1	checks for active tagging	26
6.2	Checks related to structures	27
6.3	Checks related to roles	28
6.4	Check related to mc-chunks	29
6.5	Checks related to the state of MC on a page or in a split stream	31
6.6	Benchmarks	34
II The tagpdf-user module		
Code related to L^AT_EX₂ε user commands and document commands		
Part of the tagpdf package		36
1	Setup commands	36
2	Commands related to mc-chunks	36
3	Commands related to structures	37
4	Debugging	37
5	Extension commands	38
5.1	Fake space	38
5.2	Tagging of paragraphs	38
5.3	Header and footer	39
5.4	Link tagging	39
6	Socket support	39
7	User commands and extensions of document commands	40

8	Setup and preamble commands	40
9	Commands for the mc-chunks	40
10	Commands for the structure	41
11	Socket support	42
12	Debugging	42
13	Commands to extend document commands	46
	13.1 Document structure	46
	13.2 Structure destinations	47
	13.3 Fake space	47
	13.4 Paratagging	47
	13.5 Language support	55
	13.6 Header and footer	55
	13.7 Links	57
III	The tagpdf-tree module	
	Commands trees and main dictionaries	
	Part of the tagpdf package	59
1	Trees, pdfmanagement and finalization code	59
	1.1 Check structure	59
	1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction	60
	1.3 Writing the IDtree	61
	1.4 Writing structure elements	62
	1.5 ParentTree	63
	1.6 Rolemap dictionary	66
	1.7 Classmap dictionary	66
	1.8 Namespaces	67
	1.9 Finishing the structure	68
	1.10 StructParents entry for Page	69
IV	The tagpdf-mc-shared module	
	Code related to Marked Content (mc-chunks), code shared by	
	all modes	
	Part of the tagpdf package	70
1	Public Commands	70
2	Public keys	71
3	Marked content code – shared	72
	3.1 Variables and counters	72
	3.2 Functions	73
	3.3 Keys	76

V	The tagpdf-mc-generic module	
	Code related to Marked Content (mc-chunks), generic mode	
	Part of the tagpdf package	77
1	Marked content code – generic mode	77
1.1	Variables	77
1.2	Functions	78
1.3	Looking at MC marks in boxes	81
1.4	Keys	89
VI	The tagpdf-mc-luacode module	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the tagpdf package	91
1	Marked content code – luamode code	91
1.1	Commands	93
1.2	Key definitions	97
VII	The tagpdf-struct module	
	Commands to create the structure	
	Part of the tagpdf package	100
1	Public Commands	100
2	Public keys	101
2.1	Keys for the structure commands	101
2.2	Setup keys	103
3	Variables	103
3.1	Variables used by the keys	105
3.2	Variables used by tagging code of basic elements	106
4	Commands	106
4.1	Initialization of the StructTreeRoot	107
4.2	Adding the /ID key	108
4.3	Filling in the tag info	109
4.4	Handlings kids	110
4.5	Output of the object	113
5	Keys	117
6	User commands	122
7	Attributes and attribute classes	131
7.1	Variables	131
7.2	Commands and keys	131

VIII	The tagpdf-luatex.def	
	Driver for luatex	
	Part of the tagpdf package	135
1	Loading the lua	135
2	Logging functions	139
3	Helper functions	141
	3.1 Retrieve data functions	141
	3.2 Functions to insert the pdf literals	143
4	Function for the real space chars	146
5	Function for the tagging	149
6	Parenttree	154
IX	The tagpdf-roles module	
	Tags, roles and namespace code	
	Part of the tagpdf package	156
1	Code related to roles and structure names	156
	1.1 Variables	157
	1.2 Namespaces	159
	1.3 Adding a new tag	160
	1.3.1 pdf 1.7 and earlier	161
	1.3.2 The pdf 2.0 version	163
	1.4 Helper command to read the data from files	165
	1.5 Reading the default data	167
	1.6 Parent-child rules	168
	1.6.1 Reading in the csv-files	168
	1.6.2 Retrieving the parent-child rule	170
	1.7 Remapping of tags	175
	1.8 Key-val user interface	175
X	The tagpdf-space module	
	Code related to real space chars	
	Part of the tagpdf package	178
1	Code for interword spaces	178
	Index	182

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

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

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

`activate/spaces_<key>`

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

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

Keys to activate the various tagging steps.

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

The key allows to suppress the creation of structure destinations

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

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

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

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

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

These are attributes used by the label/ref system.

1 Initialization and test if pdfmanagement is active.

```
1 <@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2024-06-20} {0.99c}
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 <*debug>
27 \ProvidesExplPackage {tagpdf-debug} {2024-06-20} {0.99c}
28   { debug code for tagpdf }
29 \@ifpackageloaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\ending}{}
30 </debug> We map the internal module name “tag” to “tagpdf” in messages.
31 <*package>
32 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
33 </package>
```

Debug mode has its special mapping:

```
32 <*debug>
33 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug } {}
34 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
35 </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} {2024-06-20} {0.99c}
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 \cs_new_protected:Npn \__tag_whatsits: {}
58 \AddToHook{begindocument}
59   {
60     \str_case:VnF \c_sys_backend_str
61     {
62       { luatex } { \cs_set_protected:Npn \__tag_whatsits: {} }
63       { dvisvgm } { \cs_set_protected:Npn \__tag_whatsits: {} }
64     }
65     {
66       \cs_set_protected:Npn \__tag_whatsits: {\tex_special:D {} }
67     }
68   }
69 </base>
```

4.1 Indexed objects

2024-04-11: Temporary code! Can be removed after the next expl3 release!

```
70 \cs_if_free:NT \pdf_object_new_indexed:nn
71   {
72     \cs_generate_variant:Nn \pdf_object_new:n {e}
73     \cs_generate_variant:Nn \pdf_object_write:nnn {enn}
74     \cs_new_protected:Npn \pdf_object_new_indexed:nn #1 #2
75     {
```



```

76     \pdf_object_new:e {#1/\int_eval:n{#2}}
77   }
78   \cs_new_protected:Npn \pdf_object_write_indexed:nnnn #1 #2 #3 #4
79     {
80       \pdf_object_write:enn {#1/\int_eval:n{#2}}{#3}{#4}
81     }
82   \cs_generate_variant:Nn \pdf_object_write_indexed:nnnn {nnne}
83   \cs_new:Npn\pdf_object_ref_indexed:nn #1 #2
84     {
85       \pdf_object_ref:e {#1/\int_eval:n{#2}}
86     }
87   \cs_new:Npn \__kernel_pdf_object_id_indexed:nn #1 #2
88     {
89       \int_use:c
90       { c__pdf_object_ #1/\int_eval:n{#2} _int }
91     }
92 }
93

```

4.2 a LastPage label

See also issue #2 in Accessible-xref

__tag_lastpagelabel:

```

94 (*package)
95 \cs_new_protected:Npn \__tag_lastpagelabel:
96   {
97     \legacy_if:nT { @filesw }
98     {
99       \exp_args:NNne \exp_args:NNe\iow_now:Nn \@auxout
100      {
101        \token_to_str:N \new@label@record
102        { @tag@LastPage }
103        {
104          {abspage} { \int_use:N \g_shipout_readonly_int }
105          {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
106          {tagstruct}{\int_use:N \c@g__tag_struct_abs_int }
107        }
108      }
109    }
110  }
111
112 \AddToHook{enddocument/afterlastpage}
113 { \__tag_lastpagelabel: }

```

(End of definition for __tag_lastpagelabel:.)

5 Variables

\l__tag_tmpa_tl A few temporary variables

```

\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

```

```

117 \tl_new:N \l__tag_get_parent_tmpa_tl
118 \tl_new:N \l__tag_get_parent_tmpb_tl
119 \str_new:N \l__tag_tmpa_str
120 \prop_new:N \l__tag_tmpa_prop
121 \seq_new:N \l__tag_tmpa_seq
122 \seq_new:N \l__tag_tmpb_seq
123 \clist_new:N \l__tag_tmpa_clist
124 \int_new:N \l__tag_tmpa_int
125 \box_new:N \l__tag_tmpa_box
126 \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
127 \clist_const:Nn \c__tag_property_mc_clist {tagabspace,tagmcabs,tagmcid}
128 \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.

```

129 \int_new:N \l__tag_loglevel_int

```

(End of definition for `\l__tag_loglevel_int`.)

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

```

130 \bool_new:N \g__tag_active_space_bool
131 \bool_new:N \g__tag_active_mc_bool
132 \bool_new:N \g__tag_active_tree_bool
133 \bool_new:N \g__tag_active_struct_bool
134 \bool_new:N \g__tag_active_struct_dest_bool
135 \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` `\l__tag_active_struct_bool` `\l__tag_active_socket_bool` These booleans should help to control the *local* behaviour of tagpdf. In some cases it could e.g. be necessary to stop tagging completely. As local booleans they respect groups. TODO: check if they are used everywhere as needed and as wanted.

```

136 \bool_new:N \l__tag_active_mc_bool
137 \bool_set_true:N \l__tag_active_mc_bool
138 \bool_new:N \l__tag_active_struct_bool
139 \bool_set_true:N \l__tag_active_struct_bool
140 \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 of empty artifact mc-chunks.

141 `\bool_new:N \g__tag_tagunmarked_bool`

(End of definition for `\g__tag_tagunmarked_bool`.)

6 Variants of l3 commands

142 `\prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F,TF}`
 143 `\cs_generate_variant:Nn \pdf_object_ref:n {e}`
 144 `\cs_generate_variant:Nn \pdfannot_dict_put:nnn {nne}`
 145 `\cs_generate_variant:Nn \pdffile_embed_stream:nnn {nee,oee}`
 146 `\cs_generate_variant:Nn \prop_gput:Nnn {Nee,Nen} %** unneeded`
 147 `\cs_generate_variant:Nn \prop_put:Nnn {Nee} %** unneeded`
 148 `\cs_generate_variant:Nn \prop_item:Nn {No,Ne} %** unneeded`
 149 `\cs_generate_variant:Nn \seq_set_split:Nnn{Nne} %** unneeded`
 150 `\cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }`
 151 `\cs_generate_variant:Nn \clist_map_inline:nn {on}`

7 Label and Reference commands

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

`__tag_property_new:nmmn` At first a command to define new properties

`__tag_property_gset:nmmn` 152 `\cs_new_eq:NN __tag_property_new:nmmn \property_new:nmmn`

`__tag_property_ref:nnn` For the non-shipout code we need also the option to reset property

153 `\cs_new_eq:NN __tag_property_gset:nmmn \property_gset:nmmn`

The command to reference while giving a local default.

154 `\cs_new_eq:NN __tag_property_ref:nnn \property_ref:nnn`

155 `\cs_new_eq:NN __tag_property_ref:nn \property_ref:nn`

The command to record

156 `\cs_new_protected:Npn __tag_property_record:nn #1#2`

```
157 {
158   \@bsphack
159   \property_record:nn{#1}{#2}
160   \@esphack
161 }
```

162

And a few variants

163 `\cs_generate_variant:Nn __tag_property_ref:nnn {enn}`

164 `\cs_generate_variant:Nn __tag_property_ref:nn {en}`

165 `\cs_generate_variant:Nn __tag_property_record:nn {en,eV}`

(End of definition for `__tag_property_new:nmmn`, `__tag_property_gset:nmmn`, 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.

```

166 \cs_new:Npn \_tag\_property\_ref\_lastpage:nn #1 #2
167 {
168   \_tag\_property\_ref:nnn {@tag@LastPage}{#1}{#2}
169 }

```

(End of definition for _tag_property_ref_lastpage:nn.)

8 Setup label attributes

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

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

```

170 \_tag\_property\_new:nnnn
171 { tagstruct } { now }
172 {0} { \int\_use:N \c@g\_tag\_struct\_abs\_int }
173 \_tag\_property\_new:nnnn { tagstructobj } { now } {}
174 {
175   \pdf\_object\_ref\_indexed:nn { \_tag/struct } { \c@g\_tag\_struct\_abs\_int }
176 }
177 \_tag\_property\_new:nnnn
178 { tagabspage } { shipout }
179 {0} { \int\_use:N \g\_shipout\_readonly\_int }
180 \_tag\_property\_new:nnnn { tagmcabs } { now }
181 {0} { \int\_use:N \c@g\_tag\_MCID\_abs\_int }
182
183 \flag\_new:n { \_tag/mcid }
184 \_tag\_property\_new:nnnn { tagmcid } { shipout }
185 {0} { \flag\_height:n { \_tag/mcid } }
186

```

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

9 Commands to fill seq and prop

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

```

\_tag\_prop\_new:N
\_tag\_prop\_new\_linked:N 187 \cs\_set\_eq:NN \_tag\_prop\_new:N \prop\_new:N
\_tag\_seq\_new:N 188 \cs\_set\_eq:NN \_tag\_prop\_new\_linked:N \prop\_new\_linked:N
\_tag\_prop\_gput:Nnn 189 \cs\_set\_eq:NN \_tag\_seq\_new:N \seq\_new:N
\_tag\_seq\_gput\_right:Nn 190 \cs\_set\_eq:NN \_tag\_prop\_gput:Nnn \prop\_gput:Nnn
\_tag\_seq\_item:cn 191 \cs\_set\_eq:NN \_tag\_seq\_gput\_right:Nn \seq\_gput\_right:Nn
\_tag\_prop\_item:cn 192 \cs\_set\_eq:NN \_tag\_seq\_item:cn \seq\_item:cn
\_tag\_seq\_show:N 193 \cs\_set\_eq:NN \_tag\_prop\_item:cn \prop\_item:cn
\_tag\_prop\_show:N 194 \cs\_set\_eq:NN \_tag\_seq\_show:N \seq\_show:N

```

```

195 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
196 % cnx temporary needed for latex-lab-graphic code
197 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nen , Nee, Nne , cnn, cen, cne, cno, cnx}
198 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Ne , No, cn, ce }
199 \cs_generate_variant:Nn \__tag_prop_new:N { c }
200 \cs_generate_variant:Nn \__tag_seq_new:N { c }
201 \cs_generate_variant:Nn \__tag_seq_show:N { c }
202 \cs_generate_variant:Nn \__tag_prop_show:N { c }
203 \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
`\tag_start:` stop the counting of paragraphs. The commands keep track of the nesting with a local
`\tag_stop:n` 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
204 \begin{package} debug
205 \int_new:N \l__tag_tag_stop_int

206 \cs_set_protected:Npn \tag_stop:
207 {
208 <debug> \msg_note:nne {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }
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:
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:nne {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }
226 }
227 \cs_set_eq:NN\tagstop\tag_stop:
228 \cs_set_eq:NN\tagstart\tag_start:

229 \cs_set_protected:Npn \tag_stop:n #1
230 {
231 <debug> \msg_note:nnee {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }{#1}

```

```

232 \int_incr:N \l__tag_tag_stop_int
233 \bool_set_false:N \l__tag_active_struct_bool
234 \bool_set_false:N \l__tag_active_mc_bool
235 \bool_set_false:N \l__tag_active_socket_bool
236 \__tag_stop_para_ints:
237 }
238 \cs_set_protected:Npn \tag_start:n #1
239 {
240 \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
241 \int_if_zero:nT { \l__tag_tag_stop_int }
242 {
243 \bool_set_true:N \l__tag_active_struct_bool
244 \bool_set_true:N \l__tag_active_mc_bool
245 \bool_set_true:N \l__tag_active_socket_bool
246 \__tag_start_para_ints:
247 }
248 <debug> \msg_note:nnee {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }{#1}
249 }
250 </package | debug>
251 <*base>
252 \cs_new_protected:Npn \tag_stop:{}
253 \cs_new_protected:Npn \tag_start:{}
254 \cs_new_protected:Npn \tagstop{}
255 \cs_new_protected:Npn \tagstart{}
256 \cs_new_protected:Npn \tag_stop:n #1 {}
257 \cs_new_protected:Npn \tag_start:n #1 {}
258 </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

Keys to (globally) activate tagging. `activate/spaces` activates the additional parsing needed for interword spaces. It is defined in `tagpdf-space`. `activate/struct-dest` allows to activate or suppress structure destinations.

`activate/mc` (setup-key)
`activate/tree` (setup-key)
`activate/struct` (setup-key)
`activate/all` (setup-key)
`activate/struct-dest` (setup-key)

```

259 <*package>
260 \keys_define:nn { __tag / setup }
261 {
262 activate/mc .bool_gset:N = \g__tag_active_mc_bool,
263 activate/tree .bool_gset:N = \g__tag_active_tree_bool,
264 activate/struct .bool_gset:N = \g__tag_active_struct_bool,
265 activate/all .meta:n =
266 {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
267 activate/all .default:n = true,
268 activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,

```

old, deprecated names

```

269 activate-mc .bool_gset:N = \g__tag_active_mc_bool,
270 activate-tree .bool_gset:N = \g__tag_active_tree_bool,
271 activate-struct .bool_gset:N = \g__tag_active_struct_bool,
272 activate-all .meta:n =

```

```

273     {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
274     activate-all .default:n = true,
275     no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,

```

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

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

```

276     debug/show .choice:,

```

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

debug/log_□(setup-key)

The log takes currently the values none, v, vv, vvv, all. The description of the log levels is in tagpdf-checks.

debug/uncompress_□(setup-key)

log_□(deprecated)

uncompress_□(deprecated)

```

277     debug/log .choice:,
278     debug/log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
279     debug/log / v .code:n =
280     {
281     \int_set:Nn \l__tag_loglevel_int { 1 }
282     \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
283     },
284     debug/log / vv .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
285     debug/log / vvv .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
286     debug/log / all .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
287     debug/uncompress .code:n = { \pdf_uncompress: },

```

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

```

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

```

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

activate/tagunmarked_□(setup-key)

tagunmarked_□(deprecated)

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

```

290     activate/tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
291     activate/tagunmarked .initial:n = true,

```

deprecated name

```

292     tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

```

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

page/tabsorder_□(setup-key)

tabsorder_□(deprecated)

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

```

293     page/tabsorder .choice:,
294     page/tabsorder / row .code:n =
295     \pdfmanagement_add:nnn { Page } {Tabs}{/R},
296     page/tabsorder / column .code:n =
297     \pdfmanagement_add:nnn { Page } {Tabs}{/C},
298     page/tabsorder / structure .code:n =
299     \pdfmanagement_add:nnn { Page } {Tabs}{/S},
300     page/tabsorder / none .code:n =
301     \pdfmanagement_remove:nn {Page} {Tabs},
302     page/tabsorder .initial:n = structure,

```

deprecated name

```
303     tabsorder .meta:n = {page/tabsorder={#1}},  
304   }
```

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

12 loading of engine/more dependent code

```
305 \sys_if_engine luatex:T  
306   {  
307     \file_input:n {tagpdf-luatex.def}  
308   }  
309 </package>  
  
310 <*mcloding>  
311 \bool_if:NTF \g__tag_mode_lua_bool  
312   {  
313     \RequirePackage {tagpdf-mc-code-lua}  
314   }  
315   {  
316     \RequirePackage {tagpdf-mc-code-generic} %  
317   }  
318 </mcloding>  
319 <*debug>  
320 \bool_if:NTF \g__tag_mode_lua_bool  
321   {  
322     \RequirePackage {tagpdf-debug-lua}  
323   }  
324   {  
325     \RequirePackage {tagpdf-debug-generic} %  
326   }  
327 </debug>
```


Part I

The `tagpdf-checks` module

Messages and check code

Part of the `tagpdf` package

1 Commands

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

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

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

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

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

2 Description of log messages

2.1 `\ShowTagging` command

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

2.2 Messages in checks and commands

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

2.3 Messages from the ptgging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

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

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

2.6 Debug mode messages and code

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

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

2.7 Messages

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

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

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

`tree-statistic` Message issued at the end of the compilation showing the number of objects to write

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

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

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

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2024-06-20} {0.99c}
4 {part of tagpdf - code related to checks, conditionals, debugging and messages}
5 </header>
```

3 Messages

3.1 Messages related to mc-chunks

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

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

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

mc-tag-missing If the tag is missing

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

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

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

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

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

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

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

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

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

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

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

mc-pushed Informational messages about mc-pushing.

mc-popped

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

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

mc-current Informational messages about current mc state.

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

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

3.2 Messages related to structures

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

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

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

struct-no-objnum Should not happen ...

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

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

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

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

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

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

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

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

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

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

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

struct-used-twice

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

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

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

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

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

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

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

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

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

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

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

tree-statistic Message issued at the end showing the estimated number of structures and MC-children

```
49 \msg_new:nnn { tag } {tree-statistic}
50 {
51   Finalizing~the~tagging~structure:\\
52   Writing~out~\c_tilde_str
53   \int_use:N\c@g__tag_struct_abs_int\c_space_tl~structure~objects\\
54   with~\c_tilde_str
55   \int_use:N\c@g__tag_MCID_abs_int\c_space_tl'MC'~leaf~nodes.\\
56   Be~patient~if~there~are~lots~of~objects!
57 }
58 </package>
```

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

The following messages are only needed in debug mode.

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

show-kids

```
59 <*debug>
60 \msg_new:nnn { tag/debug } { show-struct }
61 {
62   =====\\
63   The~structure~#1~
64   \tl_if_empty:nTF {#2}
65   { is~empty \\>~ . }
66   { contains: #2 }
67   \\
68 }
69 \msg_new:nnn { tag/debug } { show-kids }
70 {
71   The~structure~has~the~following~kids:
72   \tl_if_empty:nTF {#2}
73   { \\>~ NONE }
74   { #2 }
75   \\
76   =====
77 }
78 </debug>
```

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

3.3 Attributes

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

attr-unknown

```
79 (*package)
80 \msg_new:nnn { tag } {attr-unknown} { attribute-#1-is-unknown}
```

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

3.4 Roles

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

```
role-unknown 81 \msg_new:nnn { tag } {role-missing} { tag-#1-has-no-role-assigned }
role-unknown-tag 82 \msg_new:nnn { tag } {role-unknown} { role-#1-is-not-known }
role-unknown-NS 83 \msg_new:nnn { tag } {role-unknown-tag} { tag-#1-is-not-known }
84 \msg_new:nnn { tag } {role-unknown-NS} { \tl_if_empty:nTF{#1}{Empty-NS}{NS-#1-is-not-known}
```

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

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

```
85 \msg_new:nnn { tag } {role-parent-child}
86 { Parent-Child-’#1’--->-’#2’.\Relation-is-#3-\msg_line_context:}
```

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

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

```
87 \msg_new:nnn { tag } {role-remapping}
88 { remapping-tag-to-#1 }
```

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

role-tag Info messages.

```
new-tag 89 \msg_new:nnn { tag } {role-tag} { mapping-tag-#1-to-role-#2 }
90 \msg_new:nnn { tag } {new-tag} { adding-new-tag-#1 }
91 \msg_new:nnn { tag } {read-namespace} { reading-namespace-definitions-tagpdf-
ns-#1.def }
92 \msg_new:nnn { tag } {namespace-missing}{ namespace-definitions-tagpdf-ns-#1.def-not-found }
93 \msg_new:nnn { tag } {namespace-unknown}{ namespace-#1-is-not-declared }
```

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

3.5 Miscellaneous

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

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

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

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

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

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

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

```
98 \cs_set_eq:MN \__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.

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

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

4 Retrieving data

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

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

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

5 User conditionals

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

```
104 <*base>
105 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
106 { \prg_return_false: }
107 </base>
108 <*package>
109 \prg_set_conditional:Npnn \tag_if_active: { p , T , TF , F }
110 {
111   \bool_lazy_all:nTF
112   {
113     {\g__tag_active_struct_bool}
114     {\g__tag_active_mc_bool}
115     {\g__tag_active_tree_bool}
116     {\l__tag_active_struct_bool}
117     {\l__tag_active_mc_bool}
118   }
119   {
120     \prg_return_true:
121   }
122   {
123     \prg_return_false:
124   }
125 }
126 </package>
```

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

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

```

127 <*base>
128 \prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
129   {
130     \tl_if_exist:cTF {l_tag_box_\int_use:N #1_t1}
131     {
132       \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_t1}}>{0}
133       { \prg_return_true: }
134       { \prg_return_false: }
135     }
136     {
137       \prg_return_false:
138       % warning??
139     }
140   }
141 </base>

```

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

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

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

```

\__tag_check_if_active_struct:TF
142 <*package>
143 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
144   {
145     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
146     {
147       \prg_return_true:
148     }
149     {
150       \prg_return_false:
151     }
152   }
153 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
154   {
155     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
156     {
157       \prg_return_true:
158     }
159     {
160       \prg_return_false:
161     }
162   }

```

(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.

```

163 \cs_new_protected:Npn \_tag_check_structure_has_tag:n #1 %#1 struct num
164   {
165     \prop_if_in:cnF { g__tag_struct_#1_prop }
166       {S}
167     {
168       \msg_error:nn { tag } {struct-missing-tag}
169     }
170   }

```

(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.

```

171 \cs_new_protected:Npn \_tag_check_structure_tag:N #1
172   {
173     \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
174     {
175       \msg_warning:nne { tag } {role-unknown-tag} {#1}
176     }
177   }

```

(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.

```

178 \cs_new_protected:Npn \_tag_check_info_closing_struct:n #1 %#1 struct num
179   {
180     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
181     {
182       \msg_info:nnn { tag } {struct-show-closing} {#1}
183     }
184   }
185
186 \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.

```

187 \cs_new_protected:Npn \_tag_check_no_open_struct:
188   {
189     \msg_error:nn { tag } {struct-faulty-nesting}
190   }

```

(End of definition for _tag_check_no_open_struct:.)

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

```

191 \cs_new_protected:Npn \_tag_check_struct_used:n #1 %#1 label
192   {
193     \prop_get:cnNT
194       {g__tag_struct\_tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}

```

```

195     {P}
196     \l__tag_tmpa_tl
197     {
198         \msg_warning:nnn { tag } {struct-used-twice} {#1}
199     }
200 }

```

(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.

```

201 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
202 {
203     \tl_if_empty:nTF {#2}
204     {
205         \msg_error:nnn { tag } {role-missing} {#1}
206     }
207     {
208         \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l_tmpa_tl
209         {
210             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
211             {
212                 \msg_info:nnnn { tag } {role-tag} {#1} {#2}
213             }
214         }
215         {
216             \msg_error:nnn { tag } {role-unknown} {#2}
217         }
218     }
219 }

```

Similar with a namespace

```

220 \cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
221 {
222     \tl_if_empty:nTF {#2}
223     {
224         \msg_error:nnn { tag } {role-missing} {#1}
225     }
226     {
227         \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l_tmpa_tl
228         {
229             \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
230             {
231                 \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
232             }
233         }
234         {
235             \msg_error:nnn { tag } {role-unknown} {#2/#3}
236         }
237     }
238 }

```

(End of definition for __tag_check_add_tag_role:nn.)

6.4 Check related to mc-chunks

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

```

239 \cs_new_protected:Npn __tag_check_mc_if_nested:
240 {
241   __tag_mc_if_in:T
242   {
243     \msg_warning:nne { tag } {mc-nested} { __tag_get_mc_abs_cnt: }
244   }
245 }
246
247 \cs_new_protected:Npn __tag_check_mc_if_open:
248 {
249   __tag_mc_if_in:F
250   {
251     \msg_warning:nne { tag } {mc-not-open} { __tag_get_mc_abs_cnt: }
252   }
253 }

```

(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.

```

254 \cs_new_protected:Npn __tag_check_mc_pushed_popped:nn #1 #2
255 {
256   \int_compare:nNnT
257     { \l__tag_loglevel_int } = { 2 }
258     { \msg_info:nne {tag}{mc-#1}{#2} }
259   \int_compare:nNnT
260     { \l__tag_loglevel_int } > { 2 }
261     {
262       \msg_info:nne {tag}{mc-#1}{#2}
263       \seq_log:N \g__tag_mc_stack_seq
264     }
265 }

```

(End of definition for __tag_check_mc_pushed_popped:nn.)

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

```

266 \cs_new_protected:Npn __tag_check_mc_tag:N #1 %#1 is var with a tag name in it
267 {
268   \tl_if_empty:NT #1
269   {
270     \msg_error:nne { tag } {mc-tag-missing} { __tag_get_mc_abs_cnt: }
271   }
272   \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
273   {
274     \msg_warning:nne { tag } {role-unknown-tag} {#1}
275   }
276 }

```

(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??

```

277 \cs_new_protected:Npn \__tag_check_init_mc_used:
278   {
279     \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
280     \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
281   }

```

(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.

```

282 \cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid absent
283   {
284     \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
285     {
286       \__tag_check_init_mc_used:
287       \intarray_gset:Nnn \g__tag_check_mc_used_intarray
288         {#1}
289         { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
290       \int_compare:nNnT
291         {
292           \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
293         }
294         >
295         { 1 }
296         {
297           \msg_warning:nnn { tag } {mc-used-twice} {#1}
298         }
299     }
300   }

```

(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.

```

301 \cs_new_protected:Npn \_tag_check_show_MCID_by_page:
302   {
303     \tl_set:Ne \l__tag_tmpa_tl
304     {
305       \__tag_property_ref_lastpage:nn
306       {abspage}
307       {-1}
308     }
309     \int_step_inline:nnnn {1}{1}
310     {
311       \l__tag_tmpa_tl
312     }
313     {
314       \seq_clear:N \l_tmpa_seq
315       \int_step_inline:nnnn

```

```

316     {1}
317     {1}
318     {
319     \_tag_property_ref_lastpage:nn
320     {tagmcabs}
321     {-1}
322     }
323     {
324     \int_compare:nT
325     {
326     \_tag_property_ref:enn
327     {mcid-###1}
328     {tagabspage}
329     {-1}
330     =
331     ##1
332     }
333     {
334     \seq_gput_right:Ne \l_tmpa_seq
335     {
336     Page##1-###1-
337     \_tag_property_ref:enn
338     {mcid-###1}
339     {tagmcid}
340     {-1}
341     }
342     }
343     }
344     \seq_show:N \l_tmpa_seq
345     }
346     }

```

(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`.

```

347 \prg_new_conditional:Npnn \_tag_check_if_mc_in_galley: { T,F,TF }
348 {
349   \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
350   { \prg_return_false: }
351   { \prg_return_true: }
352 }

```

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

`_tag_check_if_mc_tmb_missing_p:` This checks if a extra top mark (“extra-tmb”) is needed. According to the analysis this
`_tag_check_if_mc_tmb_missing:TF` the case if the firstmarks start with e- or b+. Like above we assume that the marks content is already in the seq’s.

```

353 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tmb\_missing: { T,F,TF }
354 {
355   \bool\_if:nTF
356   {
357     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{e-}
358     ||
359     \str\_if\_eq\_p:ee {\seq\_item:Nn \l\_tag\_mc\_firstmarks\_seq {1}}{b+}
360   }
361   { \prg\_return\_true: }
362   { \prg\_return\_false: }
363 }

```

(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.

```

364 \prg_new_conditional:Npnn \_tag\_check\_if\_mc\_tme\_missing: { T,F,TF }
365 {
366   \str\_if\_eq:eeTF {\seq\_item:Nn \l\_tag\_mc\_botmarks\_seq {1}}{b+}
367   { \prg\_return\_true: }
368   { \prg\_return\_false: }
369 }

```

(End of definition for _tag_check_if_mc_tme_missing:TF.)

```
370 </package>
```

```
371 <*debug>
```

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

```

372 \msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl\_to\_str:n{#2}~[\msg\_l
373 \msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg\_line\_context:] }
374
375 \cs_new_protected:Npn \_tag\_debug\_mc\_begin\_insert:n #1
376 {
377   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}
378   {
379     \msg\_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
380   }
381 }
382 \cs_new_protected:Npn \_tag\_debug\_mc\_begin\_ignore:n #1
383 {
384   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}
385   {
386     \msg\_note:nnnn { tag / debug } {mc-begin} {ignored} { #1 }
387   }
388 }
389 \cs_new_protected:Npn \_tag\_debug\_mc\_end\_insert:
390 {
391   \int\_compare:nNnT { \l\_tag\_loglevel\_int } > {0}

```



```

392     {
393     \msg_note:nnn { tag / debug } {mc-end} {inserted}
394     }
395 }
396 \cs_new_protected:Npn \__tag_debug_mc_end_ignore:
397 {
398   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
399   {
400     \msg_note:nnn { tag / debug } {mc-end } {ignored}
401   }
402 }

```

And now something for the structures

```

403 \msg_new:nnn { tag / debug } {struct-begin}
404 {
405   Struct-\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\[\msg_line_context:
406 }
407 \msg_new:nnn { tag / debug } {struct-end}
408 {
409   Struct-end~#1~[\msg_line_context:]
410 }
411 \msg_new:nnn { tag / debug } {struct-end-wrong}
412 {
413   Struct-end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
414 }
415
416 \cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
417 {
418   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
419   {
420     \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
421     \seq_log:N \g__tag_struct_tag_stack_seq
422   }
423 }
424 \cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
425 {
426   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
427   {
428     \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
429   }
430 }
431 \cs_new_protected:Npn \__tag_debug_struct_end_insert:
432 {
433   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
434   {
435     \msg_note:nnn { tag / debug } {struct-end} {inserted}
436     \seq_log:N \g__tag_struct_tag_stack_seq
437   }
438 }
439 \cs_new_protected:Npn \__tag_debug_struct_end_ignore:
440 {
441   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
442   {
443     \msg_note:nnn { tag / debug } {struct-end } {ignored}
444   }

```

```

445 }
446 \cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
447 {
448   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
449   {
450     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
451     {
452       \str_if_eq:eeF
453       {#1}
454       {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
455       {
456         \msg_warning:nnee { tag/debug }{ struct-end-wrong }
457         {#1}
458         {\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl}
459       }
460     }
461   }
462 }

```

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.

```

463 \msg_new:nnn { tag / debug } {tag-stop}
464 {
465   \int_if_zero:nTF
466   {#1}
467   {Tagging~stopped}
468   {Tagging~(not)~stopped~(already~inactive)}\
469   level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
470 }
471 \msg_new:nnn { tag / debug } {tag-start}
472 {
473   \int_if_zero:nTF
474   {#1}
475   {Tagging~restarted}
476   {Tagging~(not)~restarted}\
477   level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
478 }
479 </debug>

```

6.6 Benchmarks

It doesn't make much sense to do benchmarks in debug mode or in combination with a log-level as this would slow down the compilation. So we add simple commands that can be activated if `l3benchmark` has been loaded. TODO: is a warning needed?

```

480 <*package>
481 \cs_new_protected:Npn \__tag_check_benchmark_tic:{}
482 \cs_new_protected:Npn \__tag_check_benchmark_toc:{}
483 \cs_new_protected:Npn \tag_check_benchmark_on:
484 {
485   \cs_if_exist:NT \benchmark_tic:
486   {
487     \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
488     \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:

```

```
489     }  
490   }  
491 </package>
```

Part II

The `tagpdf-user` module

Code related to L^AT_EX2e user commands and document commands Part of the `tagpdf` package

1 Setup commands

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

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

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

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

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

2 Commands related to mc-chunks

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

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

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

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

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

3 Commands related to structures

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

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

4 Debugging

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

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

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

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

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

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

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

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

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

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

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

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

5 Extension commands

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

The commands and keys should be view as experimental!

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

5.1 Fake space

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

5.2 Tagging of paragraphs

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

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

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

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

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

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

5.3 Header and footer

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

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

5.4 Link tagging

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

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

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

6 Socket support

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

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

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

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

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

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

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

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

7 User commands and extensions of document commands

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-user} {2024-06-20} {0.99c}
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 36.)

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

`\tagtool`

```

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

```

(End of definition for `\tag_tool:n` and `\tagtool`. These functions are documented on page 36.)

9 Commands for the mc-chunks

`\tagmcbegin`

`\tagmcend`

`\tagmcuse`

```

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

```



```

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

```

(End of definition for \tagmccbegin, \tagmccend, and \tagmccuse. These functions are documented on page 36.)

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

```

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

```

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

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 37.)

11 Socket support

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

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

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

```

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

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

(End of definition for `\tag_socket_use:n`, `\tag_socket_use:nn`, and `\UseTaggingSocket`. These functions are documented on page 39.)

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 37.)

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 37.)

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     \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 37.)

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

```

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

```

```

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

```

(End of definition for `mc-marks` (`show-key`). This function is documented on page 37.)

`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 37.)

`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 = 1
252 }
253 </debug>

```

(End of definition for `debug/structures (show-key)`. This function is documented on page 37.)

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 36.)

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_indexed_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 38.)

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.

<pre> \l__tag_para_bool \l__tag_para_flattened_bool \l__tag_para_show_bool \g__tag_para_begin_int \g__tag_para_end_int \g__tag_para_main_begin_int \g__tag_para_main_end_int \g__tag_para_main_struct_tl \l__tag_para_tag_default_tl \l__tag_para_tag_tl \l__tag_para_main_tag_tl \l__tag_para_attr_class_tl \l__tag_para_main_attr_class_tl</pre>	<p>At first some variables.</p> <pre> 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</pre>
--	--

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

```
300 \tl_new:N \g__tag_para_main_struct_tl
301 \tl_gset:Nn \g__tag_para_main_struct_tl {1}
302 \tl_new:N \l__tag_para_tag_default_tl
303 \tl_set:Nn \l__tag_para_tag_default_tl { text }
304 \tl_new:N \l__tag_para_tag_tl
305 \tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
306 \tl_new:N \l__tag_para_main_tag_tl
307 \tl_set:Nn \l__tag_para_main_tag_tl {text-unit}
```

this is perhaps already defined by the block code

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

(End of definition for \l__tag_para_bool and others.)

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

```
\__tag_gincr_para_main_begin_int:
\__tag_gincr_para_main_end_int:
\__tag_gincr_para_begin_int:
\__tag_gincr_para_end_int:
311 \cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
312 {
313   \int_gincr:N \g__tag_para_main_begin_int
314 }
315 \cs_new_protected:Npn \__tag_gincr_para_begin_int:
316 {
317   \int_gincr:N \g__tag_para_begin_int
318 }
319 \cs_new_protected:Npn \__tag_gincr_para_main_end_int:
320 {
321   \int_gincr:N \g__tag_para_main_end_int
322 }
323 \cs_new_protected:Npn \__tag_gincr_para_end_int:
324 {
325   \int_gincr:N \g__tag_para_end_int
326 }
```

(End of definition for __tag_gincr_para_main_begin_int: and others.)

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



```

342   {
343     \int_gincr:N \g__tag_para_end_int
344   }
345 }
346 \cs_new_protected:Npn \__tag_stop_para_ints:
347 {
348   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
349   \cs_set_eq:NN \__tag_gincr_para_begin_int: \prg_do_nothing:
350   \cs_set_eq:NN \__tag_gincr_para_main_end_int: \prg_do_nothing:
351   \cs_set_eq:NN \__tag_gincr_para_end_int: \prg_do_nothing:
352 }

```

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

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

__tag_para_main_store_struct:

```

353 \cs_new:Npn \__tag_para_main_store_struct:
354 {
355   \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
356 }

```

(End of definition for __tag_para_main_store_struct:.)

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

```

357 \AddToHook{package/latex-lab-testphase-sec/after}
358 {
359   \cs_set_protected:Npn \@kernel@tag@hangfrom #1
360   {
361     \tagstructbegin{tag=\l__tag_para_tag_tl}
362     \__tag_gincr_para_begin_int:
363     \tagstructbegin{tag=Lbl}
364     \setbox\@tempboxa
365     \hbox
366     {
367       \bool_lazy_and:nnT
368       {\tag_if_active_p:}
369       {\g__tag_mode_lua_bool}
370       {\tagmcbegin{tag=Lbl}}
371       {#1}
372     }
373     \tag_stop:n{hangfrom}
374     \hangindent \wd\@tempboxa\noindent
375     \tag_start:n{hangfrom}
376     \tagmcbegin{\box\@tempboxa\tagmcbegin\tagstructend\tagmcbegin{}}
377   }
378 }

```

and one temporary adaptations for the block module:

```

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

```

```
385 }
386
```

```
para/tagging_␣(setup-key) These keys enable/disable locally paratagging. Paragraphs are typically tagged with two
para/tag_␣(setup-key) structure: A main structure around the whole paragraph, and inner structures around
para/maintag_␣(setup-key) the various chunks. Debugging can be activated locally with debug/show=para, this can
para/tagging_␣(tool-key) affect the typesetting as the small numbers are boxes and they have a (small) height.
para/tag_␣(tool-key) Debugging can be deactivated with debug/show=paraOff The para/tag key sets the tag
para/maintag_␣(tool-key) used by the inner structure, para/maintag the tag of the outer structure, both can also
para/flattened_␣(tool-key) be changed with \tag_tool:n
unittag_␣(deprecated)
para-flattened_␣(deprecated)
paratagging_␣(deprecated)
paratagging-show_␣(deprecated)
paratag_␣(deprecated)
387 \keys_define:nn { __tag / setup }
388 {
389   para/tagging .bool_set:N = \l__tag_para_bool,
390   debug/show/para .code:n = {\bool_set_true:N \l__tag_para_show_bool},
391   debug/show/paraOff .code:n = {\bool_set_false:N \l__tag_para_show_bool},
392   para/tag .tl_set:N = \l__tag_para_tag_tl,
393   para/maintag .tl_set:N = \l__tag_para_main_tag_tl,
394   para/flattened .bool_set:N = \l__tag_para_flattened_bool
395 }
396 \keys_define:nn { tag / tool}
397 {
398   para/tagging .bool_set:N = \l__tag_para_bool,
399   para/tag .tl_set:N = \l__tag_para_tag_tl,
400   para/maintag .tl_set:N = \l__tag_para_main_tag_tl,
401   para/flattened .bool_set:N = \l__tag_para_flattened_bool
402 }
```

the deprecated names

```
403 \keys_define:nn { __tag / setup }
404 {
405   paratagging .bool_set:N = \l__tag_para_bool,
406   paratagging-show .bool_set:N = \l__tag_para_show_bool,
407   paratag .tl_set:N = \l__tag_para_tag_tl
408 }
409 \keys_define:nn { tag / tool}
410 {
411   para .bool_set:N = \l__tag_para_bool,
412   paratag .tl_set:N = \l__tag_para_tag_tl,
413   unittag .tl_set:N = \l__tag_para_main_tag_tl,
414   para-flattened .bool_set:N = \l__tag_para_flattened_bool
415 }
```

(End of definition for para/tagging (setup-key) and others. These functions are documented on page 38.)

Helper command for debugging:

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

```

425 }
426
427 \cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
428 %#1 color, #2 prefix
429 {
430   \bool_if:NT \l__tag_para_show_bool
431   {
432     \tag_mc_begin:n{artifact}
433     \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
434     \tag_mc_end:
435   }
436 }

```

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

```

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

```

```

473         {
474             tag=\l__tag_para_main_tag_tl,
475             attribute-class=\l__tag_para_main_attr_class_tl,
476         }
477         \__tag_para_main_store_struct:
478     }
479 }
480 \__tag_gincr_para_begin_int:
481 \__tag_check_typeout_v:n {==>~increment~ P \on@line }
482 \tag_struct_begin:n
483 {
484     tag=\l__tag_para_tag_tl
485     ,attribute-class=\l__tag_para_attr_class_tl
486 }
487 \__tag_check_para_begin_show:nn {green}{\PARALABEL}
488 \tag_mc_begin:n {}
489 }
490 }
491 }

```

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

```

492 \socket_new_plug:nnn{tagsupport/para/end}{plain}
493 {
494     \bool_if:NT \l__tag_para_bool
495     {
496         \__tag_gincr_para_end_int:
497         \__tag_check_typeout_v:n {==>~increment~ /P \on@line }
498         \tag_mc_end:
499         \__tag_check_para_end_show:nn {red}{}
500         \tag_struct_end:
501         \bool_if:NF \l__tag_para_flattened_bool
502         {
503             \__tag_gincr_para_main_end_int:
504             \tag_struct_end:
505         }
506     }
507 }

```

By default we assign the plain plug:

```

508 \socket_assign_plug:nn { tagsupport/para/begin}{plain}
509 \socket_assign_plug:nn { tagsupport/para/end}{plain}

```

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

```

510 \AddToHook{para/begin}{ \socket_use:n { tagsupport/para/begin }
511 }
512 \AddToHook{para/end} { \socket_use:n { tagsupport/para/end } }

```

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

```

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

```

```

517 \AddToHook{para/begin}[tagpdf]
518 {
519   \socket_use:n { tagsupport/para/begin }
520 }
521 \AddToHook{para/end}[tagpdf]
522 {
523   \socket_use:n { tagsupport/para/end }
524 }
525 \socket_assign_plug:nn { tagsupport/para/begin}{block}
526 }
527

```

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.

```

528 \AddToHook{enddocument/info}
529 {
530   \tag_if_active:F
531   {
532     \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
533   }
534   \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
535   {
536     \msg_error:nneee
537     {tag}
538     {para-hook-count-wrong}
539     {\int_use:N\g__tag_para_main_begin_int}
540     {\int_use:N\g__tag_para_main_end_int}
541     {text-unit}
542   }
543   \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
544   {
545     \msg_error:nneee
546     {tag}
547     {para-hook-count-wrong}
548     {\int_use:N\g__tag_para_begin_int}
549     {\int_use:N\g__tag_para_end_int}
550     {text}
551   }
552 }

```

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

```

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

```

```

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

```

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

`\tagpdfparaOff`

```

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

```

(End of definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 38.)

`\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
}
{\tagpdfsuppressmarks{\tagmccend}\tagstructend}%

595 \NewDocumentCommand\tagpdfsuppressmarks{m}
596   {{\use:c{__tag_mc_disable_marks:} #1}}

```

(End of definition for `\tagpdfsuppressmarks`. This function is documented on page 38.)

13.5 Language support

With the following key the lang variable is set. All structures in the current group will then set this lang variable.

test/lang_□(setup-key)

```
597 \keys_define:nn { __tag / setup }
598 {
599   text / lang .tl_set:N = \l__tag_struct_lang_tl
600 }
```

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

13.6 Header and footer

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

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

```

637     {
638       \tag_mc_begin_pop:n{ }
639     }
640     {
641       \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
642     }
643   }

```

This version allows to use an Artifact structure

```

644 \__tag_attr_new_entry:nn {__tag/attr/pagination}{/0/Artifact/Type/Pagination}
645 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
646 {
647   \bool_set_false:N \l__tag_para_bool
648   \bool_if:NTF \g__tag_mode_lua_bool
649   {
650     \tag_mc_end_push:
651   }
652   {
653     \bool_gset_eq:NN \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
654     \bool_gset_false:N \g__tag_in_mc_bool
655   }
656   \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
657   \tag_mc_begin:n {artifact=#1}
658   \tag_stop:n{headfoot}
659 }
660
661 \cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
662 {
663   \tag_start:n{headfoot}
664   \tag_mc_end:
665   \tag_struct_end:
666   \bool_if:NTF \g__tag_mode_lua_bool
667   {
668     \tag_mc_begin_pop:n{ }
669   }
670   {
671     \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
672   }
673 }

```

And now the keys

[page/exclude-header-footer](#)_□(setup-key)

[exclude-header-footer](#)_□(deprecated)

```

674 \keys_define:nn { __tag / setup }
675 {
676   page/exclude-header-footer .choice:,
677   page/exclude-header-footer / true .code:n =
678   {
679     \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
680     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
681     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_headfoot_end:
682     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_headfoot_end:
683   },
684   page/exclude-header-footer / pagination .code:n =
685   {

```



```

686     \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
687     \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
688     \cs_set_eq:NN \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
689     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \__tag_exclude_struct_headfoot_end:
690   },
691   page/exclude-header-footer / false .code:n =
692   {
693     \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
694     \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
695     \cs_set_eq:NN \__tag_hook_kernel_after_head: \prg_do_nothing:
696     \cs_set_eq:NN \__tag_hook_kernel_after_foot: \prg_do_nothing:
697   },
698   page/exclude-header-footer .default:n = true,
699   page/exclude-header-footer .initial:n = true,

```

deprecated name

```

700   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
701 }

```

(End of definition for page/exclude-header-footer (setup-key) and exclude-header-footer (deprecated). These functions are documented on page 39.)

13.7 Links

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

```

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

```

```

729     \tag_mc_end_push:
730     \tag_struct_begin:n{tag=Link}
731     \tag_mc_begin:n{tag=Link}
732     \pdfannot_dict_put:nne
733     { link/GoTo }
734     { StructParent }
735     { \tag_struct_parent_int: }
736   }
737
738   \hook_gput_code:nnn
739   {pdfannot/link/GoTo/after}
740   {tagpdf}
741   {
742     \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
743     \tag_mc_end:
744     \tag_struct_end:
745     \tag_mc_begin_pop:n{ }
746   }
747 }
748
749 % "alternative descriptions " for PAX3. How to get better text here??
750 \pdfannot_dict_put:nnn
751 { link/URI }
752 { Contents }
753 { (url) }
754
755 \pdfannot_dict_put:nnn
756 { link/GoTo }
757 { Contents }
758 { (ref) }
759
</package>

```

Part III

The tagpdf-tree module

Commands trees and main dictionaries

Part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-tree-code} {2024-06-20} {0.99c}
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   \msg_note:nn {tag}{tree-statistic}
29 }
```

(End of definition for __tag_tree_final_checks:.)

1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction

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

```

__tag/struct/1 This is the object for the root object, the StructTreeRoot
30 \pdf_object_new_indexed:nm { __tag/struct }{ 1 }
(End of definition for __tag/struct/1.)

```

```

\g__tag_tree_openaction_struct_tl We need a variable that indicates which structure is wanted in the OpenAction. By
default we use 2 (the Document structure).
31 \tl_new:N \g__tag_tree_openaction_struct_tl
32 \tl_gset:Nn \g__tag_tree_openaction_struct_tl { 2 }
(End of definition for \g__tag_tree_openaction_struct_tl.)

```

```

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

```

```

33 \keys_define:nm { __tag / setup }
34 {
35   viewer/startstructure .code:n =
36   {
37     \tl_gset:Ne \g__tag_tree_openaction_struct_tl {#1}
38   }
39   ,viewer/startstructure .default:n = { \int_use:N \c@g__tag_struct_abs_int }
40 }

```

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

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

```

41 \cs_new_protected:Npn \__tag_tree_update_openaction:
42 {
43   \prop_get:cnNT
44   { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog } }
45   {OpenAction}
46   \l__tag_tmpa_tl
47   {

```

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

```

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

```

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

```

57             \int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1}
58             { /\seq_item:Nn\l__tag_tmpa_seq{2} }
59             { ] }
60
61             >>
62         }
63     }
64 }
65 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
66 {
67     \bool_if:NT \g__tag_active_tree_bool
68     {
69         \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
70         \pdfmanagement_add:nne
71         { Catalog }
72         { StructTreeRoot }
73         { \pdf_object_ref_indexed:nn { __tag/struct } { 1 } }
74         \__tag_tree_update_openaction:
75     }
76 }

```

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

```

77 \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
78 \cs_generate_variant:Nn \tl_count:n {e}
79 \hook_gput_code:nnn{begindocument}{tagpdf}
80 {
81     \int_gset:Nn\g__tag_tree_id_pad_int
82     {\tl_count:e { \__tag_property_ref_lastpage:nn{tagstruct}{1000}}+1}
83 }
84

```

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.

```

85 \cs_new_protected:Npn \__tag_tree_write_idtree:
86 {
87     \tl_clear:N \l__tag_tmpa_tl
88     \tl_clear:N \l__tag_tmpb_tl
89     \int_zero:N \l__tag_tmpa_int
90     \int_step_inline:nnn {2} {\c@g__tag_struct_abs_int}
91     {
92         \int_incr:N\l__tag_tmpa_int
93         \tl_put_right:Ne \l__tag_tmpa_tl

```

```

94     {
95       \__tag_struct_get_id:n{##1}~\pdf_object_ref_indexed:nn {__tag/struct}{##1}~
96     }
97   \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
98   {
99     \pdf_object_unnamed_write:ne {dict}
100     { /Limits~[\__tag_struct_get_id:n{##1}~\l__tag_tmpa_int+1}~\__tag_struct_get_id:
101       /Names~[\l__tag_tmpa_tl]
102     }
103     \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
104     \int_zero:N \l__tag_tmpa_int
105     \tl_clear:N \l__tag_tmpa_tl
106   }
107 }
108 \tl_if_empty:NF \l__tag_tmpa_tl
109 {
110   \pdf_object_unnamed_write:ne {dict}
111   {
112     /Limits~
113     [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int~\l__tag_tmpa_int+1}~
114     \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
115     /Names~[\l__tag_tmpa_tl]
116   }
117   \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
118 }
119 \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
120 \__tag_prop_gput:cne
121   { g__tag_struct_1_prop }
122   { IDTree }
123   { \pdf_object_ref_last: }
124 }

```

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.

```

125 \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
126   {
127     \__tag_prop_gput:cne
128     { g__tag_struct_1_prop }
129     { ParentTree }
130     { \pdf_object_ref:n { __tag/tree/parenttree } }
131     \__tag_prop_gput:cne
132     { g__tag_struct_1_prop }
133     { RoleMap }
134     { \pdf_object_ref:n { __tag/tree/rolemap } }
135     \__tag_struct_fill_kid_key:n { 1 }
136     \prop_gremove:cn { g__tag_struct_1_prop } {S}
137     \__tag_struct_get_dict_content:nN { 1 } \l__tag_tmpa_tl
138     \pdf_object_write_indexed:nnne
139     { __tag/struct } { 1 }
140     {dict}
141     {

```

```

142         \l__tag_tmpa_tl
143     }

```

Better put S back, see <https://github.com/latex3/tagging-project/issues/86>

```

144     \prop_gput:cnn { g__tag_struct_1_prop } {S}{ /StructTreeRoot }
145 }

```

(End of definition for `__tag_tree_write_structtreeroot:`.)

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

```

146 \cs_new_protected:Npn \__tag_tree_write_structelems:
147 {
148     \int_step_inline:nnnn {2}{1}{\c@g__tag_struct_abs_int}
149     {
150         \__tag_struct_write_obj:n { ##1 }
151     }
152 }

```

(End of definition for `__tag_tree_write_structelems:`.)

1.5 ParentTree

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

```

153 \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`.

```

154 \newcounter { g__tag_parenttree_obj_int }
155 \hook_gput_code:nnn{begindocument}{tagpdf}
156 {
157     \int_gset:Nn
158         \c@g__tag_parenttree_obj_int
159         { \__tag_property_ref_lastpage:nn{abspage}{100} }
160 }

```

(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`

```

161 \tl_new:N \g__tag_parenttree_objr_tl

```

(End of definition for `\g__tag_parenttree_objr_tl:`.)

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

```

162 \cs_new_protected:Npn \_tag_parenttree_add_objr:mn #1 #2 %#1 StructParent number, #2 objref
163 {
164   \tl_gput_right:Ne \g__tag_parenttree_objr_tl
165     {
166       #1 \c_space_tl #2 ^^J
167     }
168 }

```

(End of definition for `_tag_parenttree_add_objr:mn`.)

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

```

169 \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.

```

170 \cs_new_protected:Npn \_tag_tree_parenttree_rerun_msg: {}
171 \cs_new_protected:Npn \_tag_tree_fill_parenttree:
172 {
173   \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clear
174     { %page ##1
175       \prop_clear:N \l__tag_tmpa_prop
176       \int_step_inline:nnnn{1}{1}{\_tag_property_ref_lastpage:nn{tagmcabs}{-
177         1}}
178         {
179           %mcid###1
180           \int_compare:nT
181             {\_tag_property_ref:enn{mcid-###1}{tagabspage}{-1}=##1} %mcid is on current p
182             {% yes
183               \prop_put:Nee
184                 \l__tag_tmpa_prop
185                 {\_tag_property_ref:enn{mcid-###1}{tagmcid}{-1}}
186                 {\prop_item:Nn \g__tag_mc_parenttree_prop {###1}}
187             }
188           \tl_put_right:Ne\l__tag_parenttree_content_tl
189             {
190               \int_eval:n {##1-1}\c_space_tl
191               [\c_space_tl %]
192             }
193           \int_step_inline:nnnn %###1
194             {0}
195             {1}
196             { \prop_count:N \l__tag_tmpa_prop -1 }
197             {
198               \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
199               {% page#1:mcid##1:\l__tag_tmpa_tl :content
200                 \tl_put_right:Ne \l__tag_parenttree_content_tl
201                   {
202                     \prop_if_exist:cTF { g__tag_struct_ \l__tag_tmpa_tl _prop }
203                     {

```



```

204         \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_tmpa_tl }
205     }
206     {
207         null
208     }
209     \c_space_tl
210 }
211 }
212 {
213     \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
214     {
215         \msg_warning:nn { tag } {tree-mcid-index-wrong}
216     }
217 }
218 }
219 \tl_put_right:Nn
220 \l__tag_parenttree_content_tl
221 {%[
222 ]^^J
223 }
224 }
225 }

```

(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.

```

226 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
227 {
228     \tl_set:Nn \l__tag_parenttree_content_tl
229     {
230         \lua_now:e
231         {
232             ltx.__tag.func.output_parenttree
233             (
234                 \int_use:N\g_shipout_readonly_int
235             )
236         }
237     }
238 }

```

(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?

```

239 \cs_new_protected:Npn \__tag_tree_write_parenttree:
240 {
241     \bool_if:NTF \g__tag_mode_lua_bool
242     {
243         \__tag_tree_lua_fill_parenttree:
244     }
245     {
246         \__tag_tree_fill_parenttree:
247     }
248     \__tag_tree_parenttree_rerun_msg:

```

```

249 \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
250 \pdf_object_write:nne { __tag/tree/parenttree }{dict}
251 {
252   /Nums\c_space_tl [\l__tag_parenttree_content_tl]
253 }
254 }

```

(End of definition for __tag_tree_write_parenttree:.)

1.6 Rolemap dictionary

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

`__tag/tree/rolemap` At first we reserve again an object. Rolemap is also used in PDF 2.0 as a fallback.

```

255 \pdf_object_new:n { __tag/tree/rolemap }

```

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

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

```

256 \cs_new_protected:Npn \__tag_tree_write_rolemap:
257 {
258   \bool_if:NT \g__tag_role_add_mathml_bool
259   {
260     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
261     {
262       \prop_gput:Nnn \g__tag_role_rolemap_prop {##1}{Span}
263     }
264   }
265   \prop_map_inline:Nn\g__tag_role_rolemap_prop
266   {
267     \tl_if_eq:nnF {##1}{##2}
268     {
269       \pdfdict_gput:nne {g__tag_role/RoleMap_dict}
270       {##1}
271       {\pdf_name_from_unicode_e:n{##2}}
272     }
273   }
274   \pdf_object_write:nne { __tag/tree/rolemap }{dict}
275   {
276     \pdfdict_use:n{g__tag_role/RoleMap_dict}
277   }
278 }

```

(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:
```

```
279 \cs_new_protected:Npn __tag_tree_write_classmap:
280 {
281   \tl_clear:N \l__tag_tmpa_tl
282   \seq_map_inline:Nn \g__tag_attr_class_used_seq
283   {
284     \prop_gput:Nnn \g__tag_attr_class_used_prop {##1}{ }
285   }
286   \prop_map_inline:Nn \g__tag_attr_class_used_prop
287   {
288     \tl_put_right:Ne \l__tag_tmpa_tl
289     {
290       ##1\c_space_tl
291       <<
292       \prop_item:Nn
293       \g__tag_attr_entries_prop
294       {##1}
295       >>
296       \iow_newline:
297     }
298   }
299   \tl_if_empty:NF
300   \l__tag_tmpa_tl
301   {
302     \pdf_object_new:n { __tag/tree/classmap }
303     \pdf_object_write:nne
304     { __tag/tree/classmap }
305     {dict}
306     { \l__tag_tmpa_tl }
307     \__tag_prop_gput:cne
308     { g__tag_struct_1_prop }
309     { ClassMap }
310     { \pdf_object_ref:n { __tag/tree/classmap } }
311   }
312 }
```

(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
```

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

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

```
\__tag_tree_write_namespaces:
```

```
314 \cs_new_protected:Npn __tag_tree_write_namespaces:
315 {
316   \pdf_version_compare:NnF < {2.0}
```

```

317 {
318   \prop_map_inline:Nn \g__tag_role_NS_prop
319   {
320     \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
321     {
322       \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
323       {
324         \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
325         }
326       \pdfdict_gput:nne{g__tag_role/namespace_##1_dict}
327       {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
328     }
329     \pdf_object_write:nne{tag/NS/##1}{dict}
330     {
331       \pdfdict_use:n {g__tag_role/namespace_##1_dict}
332     }
333   }
334   \pdf_object_write:nne {__tag/tree/namespaces}{array}
335   {
336     \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:n}
337   }
338 }
339 }

```

(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:

```

340 \hook_new:n {tagpdf/finish/before}
341 \cs_new_protected:Npn \__tag_finish_structure:
342 {
343   \bool_if:NT\g__tag_active_tree_bool
344   {
345     \hook_use:n {tagpdf/finish/before}
346     \__tag_tree_final_checks:
347     \iow_term:n{Package~tagpdf~Info:~writing~ParentTree}
348     \__tag_check_benchmark_tic:
349     \__tag_tree_write_parenttree:
350     \__tag_check_benchmark_toc:
351     \iow_term:n{Package~tagpdf~Info:~writing~IDTree}
352     \__tag_check_benchmark_tic:
353     \__tag_tree_write_idtree:
354     \__tag_check_benchmark_toc:
355     \iow_term:n{Package~tagpdf~Info:~writing~RoleMap}
356     \__tag_check_benchmark_tic:
357     \__tag_tree_write_rolemap:
358     \__tag_check_benchmark_toc:
359     \iow_term:n{Package~tagpdf~Info:~writing~ClassMap}
360     \__tag_check_benchmark_tic:
361     \__tag_tree_write_classmap:

```

```

362     \__tag_check_benchmark_toc:
363     \iow_term:n{Package~tagpdf~Info:~writing~NameSpaces}
364     \__tag_check_benchmark_tic:
365     \__tag_tree_write_namespaces:
366     \__tag_check_benchmark_toc:
367     \iow_term:n{Package~tagpdf~Info:~writing~StructElems}
368     \__tag_check_benchmark_tic:
369     \__tag_tree_write_structelements: %this is rather slow!!
370     \__tag_check_benchmark_toc:
371     \iow_term:n{Package~tagpdf~Info:~writing~Root}
372     \__tag_check_benchmark_tic:
373     \__tag_tree_write_structtreeroot:
374     \__tag_check_benchmark_toc:
375   }
376 }
377 </package>

```

(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.

```

378 <*package>
379 \hook_gput_code:nnn{begindocument}{tagpdf}
380 {
381   \bool_if:NT\g__tag_active_tree_bool
382   {
383     \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
384     {
385       \pdfmanagement_add:nne
386       { Page }
387       { StructParents }
388       { \int_eval:n { \g_shipout_readonly_int} }
389     }
390   }
391 }
392 </package>

```

Part IV

The `tagpdf-mc-shared` module

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

Part of the `tagpdf` package

1 Public Commands

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

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

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

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

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

New: 2019-11-20

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

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

New: 2021-04-22

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

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

`\tag_mc_reset_box:N` \star `\tag_mc_reset_box:N` $\{<box>\}$

New: 2023-06-11

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

2 Public keys

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

`tag□(mc-key)`

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

`artifact□(mc-key)`

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

`raw□(mc-key)`

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

`alt□(mc-key)`

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

`actualtext□(mc-key)`

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

`label□(mc-key)`

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

`stash□(mc-key)`

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

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

3 Marked content code – shared

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2024-06-20} {0.99c}
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 `\c1@ckpt` and restored e.g. in tabulars and align. `\int_new:N \c@g_@@_MCID_abs_int` and `\tl_put_right:Nn\c1@ckpt{\@elt{g_@@_MCID_abs_int}}` would work too, but as the name is not `expl3` then too, why bother? The absolute counter can be used to label and to check if the page counter needs a reset.

`g__tag_MCID_abs_int`

```

7 <*base>
8 \newcounter { g__tag_MCID_abs_int }

```

(End of definition for g__tag_MCID_abs_int.)

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

```

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

```

(End of definition for __tag_get_data_mc_counter:.)

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

```

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

```

(End of definition for __tag_get_mc_abs_cnt:.)

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

```

16 \bool_new:N \g__tag_in_mc_bool

```

(End of definition for \g__tag_in_mc_bool.)

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

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

```

17 \__tag_prop_new_linked:N \g__tag_mc_parenttree_prop

```

(End of definition for \g__tag_mc_parenttree_prop.)

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

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

(End of definition for `\g__tag_mc_parenttree_prop`.)

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

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

(End of definition for `\l__tag_mc_artifact_type_tl`.)

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

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

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

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

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

```
\g__tag_mc_key_tag_tl
```

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

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

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

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

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

3.2 Functions

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

tagabspage: the absolute page, `\g_shipout_readonly_int`,

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

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

```
27 {
```

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

```
29 }
```

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

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

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

```
31 {
```

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

```
33 }
```

```
34 </shared>
```

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

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

TODO: is testing for struct the right test?

```

35 <base>\cs_new_protected:Npn \tag_mc_use:n #1 { \__tag_whatsits: }
36 <*shared>
37 \cs_set_protected:Npn \tag_mc_use:n #1 %#1: label name
38   {
39     \__tag_check_if_active_struct:T
40     {
41       \tl_set:Nc \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 70.)

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

```

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

```

(End of definition for `\tag_mc_artifact_group_begin:n` and `\tag_mc_artifact_group_end:.` These functions are documented on page 70.)

`\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 71.)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

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

(End of definition for `\tag_mc_end_push:` and `\tag_mc_begin_pop:n`. These functions are documented on page 70.)

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:n { __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 71.)

```
159 </shared>
```

Part V

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

1 Marked content code – generic mode

```
1 <@@=tag>
2 <*generic>
3 \ProvidesExplPackage {tagpdf-mc-code-generic} {2024-06-20} {0.99c}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-generic} {2024-06-20} {0.99c}
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
`\g__tag_mc_footnote_marks_seq` previous chunk in the stream. We provide three by default: main, footnote and multicol.
`\g__tag_mc_multicol_marks_seq` TODO: perhaps an interface for more streams will be needed.

```
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, 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 set two marks to be able to detect the case when no mark is on a page/galley. MC-begin commands will set (b,-,data) and (b+,data), MC-end commands will set (e,-,data) and (e+,data).

```

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-
90   num
91   {
92     %\prop_show:N \g__tag_struct_cont_mc_prop
93     \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
94     {

```

```

94     \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_c
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 and is currently either `main` for the main galley, `footnote` for footnote text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

`__tag_add_missing_mcs_to_stream:Nn`

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

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

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

```

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

```

First set up a temp box for trial splitting.

```

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

```

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

```

190     \vbox_set_split_to_ht:NNn \l__tag_tmpa_box \l__tag_tmpa_box \c_max_dim

```

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

```

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

```

Some debugging info:

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

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

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

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

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

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

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

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

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

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

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

`__tag_mc_if_in:TF`
`\tag_mc_if_in_p:`
`\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 70.)

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

```

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

```

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

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

```

234 \bool_if:NTF \g__tag_delayed_shipout_bool
235   {
236     \hook_gput_code:nnn {shipout/before}{tagpdf}{ \flag_clear:n { __tag/mcid } }
237     \cs_set_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
238       {
239         \int_gincr:N \c@g__tag_MCID_abs_int
240         \__tag_property_record:eV
241         {
242           mcid-\int_use:N \c@g__tag_MCID_abs_int
243         }
244         \__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:nmmn {tagmcid } { now }
266   {0} { \int_use:N \g__tag_MCID_tmp_bypage_int }
267 \cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
268   {
269     \int_gincr:N \c@g__tag_MCID_abs_int
270     \tl_set:Ne \l__tag_mc_ref_abspage_tl
271     {
272       \__tag_property_ref:enn %3 args
273       {
274         mcid-\int_use:N \c@g__tag_MCID_abs_int
275       }
276       { tagabspage }
277       {-1}
278     }
279     \prop_get:NoNTF
280     \g__tag_MCID_byabspage_prop
281     {
282       \l__tag_mc_ref_abspage_tl
283     }
284     \l__tag_mc_tmpa_tl
285     {
286       %key already present, use value for MCID and add 1 for the next
287       \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
288       \__tag_prop_gput:Nee
289       \g__tag_MCID_byabspage_prop
290       { \l__tag_mc_ref_abspage_tl }
291       { \int_eval:n { \l__tag_mc_tmpa_tl +1 } }
292     }
293     {
294       %key not present, set MCID to 0 and insert 1
295       \int_gzero:N \g__tag_MCID_tmp_bypage_int
296       \__tag_prop_gput:Nee
297       \g__tag_MCID_byabspage_prop
298       { \l__tag_mc_ref_abspage_tl }
299       { 1 }
300     }
301     \__tag_property_record:eV
302     {
303       mcid-\int_use:N \c@g__tag_MCID_abs_int
304     }
305     \c__tag_property_mc_clist
306     \__tag_mc_bdc:ne
307     { #1 }

```

```

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

```

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

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

__tag_mc_handle_stash:e

```

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

```

(End of definition for __tag_mc_handle_stash:n.)

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

__tag_mc_bmc_artifact:n
 __tag_mc_handle_artifact:N

```

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

```

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

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

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

(End of definition for `__tag_get_data_mc_tag:.`)

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

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

set default MC tags to structure:

```
370   \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
371   \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
372   \keys_set:nn { __tag / mc } {#1}
373   \bool_if:NTF \l__tag_mc_artifact_bool
374   { %handle artifact
375     \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
376     \exp_args:NV
377     \__tag_mc_artifact_begin_marks:n \l__tag_mc_artifact_type_tl
378   }
379   { %handle mcid type
380     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
381     \__tag_mc_handle_mcid:VV
382     \l__tag_mc_key_tag_tl
383     \l__tag_mc_key_properties_tl
384     \__tag_mc_begin_marks:oo{\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 70.)

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:Ne \l__tag_mc_key_tag_tl { #1 }
459     \tl_gset:Ne \g__tag_mc_key_tag_tl { #1 }
460   },
461   raw .code:n =
462   {
463     \tl_put_right:Ne \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 71.)

Part VI

The `tagpdf-mc-luacode` module Code related to Marked Content (mc-chunks), luamode-specific Part of the `tagpdf` package

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

1 Marked content code – luamode code

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

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

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

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

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

`tag` : the type (a string)

`raw` : more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2024-06-20} {0.99c}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
6 <*debug>
7 \ProvidesExplPackage {tagpdf-debug-lua} {2024-06-20} {0.99c}
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.

```

66 \prg_new_conditional:Nnn \_tag_mc_if_in: {p,T,F,TF}

```

```

67 {
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__tag_mc_if_in:))
74      }
75     }
76     { \prg_return_false: }
77     { \prg_return_true: }
78 }

```

```

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 70.)

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

```

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

```

```

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

```

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

`__tag_mc_insert_mcid_kids:n` These commands will in the finish code replace the dummy for a mc by the real mcid
`__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_mccabs
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 70.)

\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 70.)

`\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 71.)

`__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:Ne \l__tag_mc_key_tag_tl { #1 }
260     \tl_gset:Ne \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:Ne \l__tag_mc_key_properties_tl { #1 }
269     \lua_now:e
270     {
271       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw", "#1")
272     }
273   },
274   alt .code:n      = % Alt property
275   {
276     \tl_if_empty:oF{#1}
277     {
278       \str_set_convert:Noon
279       \l__tag_tmpa_str
280       { #1 }
281       { default }
282       { utf16/hex }
283       \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }

```

```

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

```

```

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

```

(End of definition for tag (mc-key) and others. These functions are documented on page 71.)

Part VII

The tagpdf-struct module

Commands to create the structure Part of the tagpdf package

1 Public Commands

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

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

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

These commands insert a structure previously stashed away as kid into the currently active structure. A structure should be used only once, if the structure already has a parent a warning is issued.

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

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

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

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

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

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

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

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

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

2 Public keys

2.1 Keys for the structure commands

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

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

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

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

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

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

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

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

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

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

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

The value `AF-inline` is some text, which is embedded in the PDF as a text file with mime type `text/plain`. `AF-inline-o` is like `AF-inline` but expands the value once.

Future versions will perhaps extend this to more mime types, but it is still a research task to find out what is really needed.

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

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

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

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

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

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

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

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

attribute-class_□(struct-key)

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

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

2.2 Setup keys

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

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

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2024-06-20} {0.99c}
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_gset:Nn \c@g__tag_struct_abs_int { 1 }
```

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

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

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

(End of definition for `\g__tag_struct_objR_seq`.)

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

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

(End of definition for `\c__tag_struct_null_tl`.)

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

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

(End of definition for `\g__tag_struct_cont_mc_prop`.)

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

```
12 \seq_new:N \g__tag_struct_stack_seq
13 \seq_gpush:Nn \g__tag_struct_stack_seq {1}
```

(End of definition for `\g__tag_struct_stack_seq`.)

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

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

(End of definition for `\g__tag_struct_tag_stack_seq`.)

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

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

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

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

The data of the StructTreeRoot and the StructElem are in properties: `\g_@@_struct_1_prop` for the root and `\g_@@_struct_N_prop`, $N \geq 2$ for the other.

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

All properties have at least the keys

Type StructTreeRoot or StructElem

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

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

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

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

3.1 Variables used by the keys

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

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

```

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

```

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

```

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

```

(End of definition for `\l__tag_struct_key_label_tl`.)

```

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

```

(End of definition for `\l__tag_struct_elem_stash_bool`.)

3.2 Variables used by tagging code of basic elements

```

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

```

```

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

```

(End of definition for `\g__tag_struct_dest_num_prop`.)

```

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

```

```

67 \prop_new_linked:N \g__tag_struct_ref_by_dest_prop

```

(End of definition for `\g__tag_struct_ref_by_dest_prop`.)

4 Commands

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

```

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

```

```

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

```

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

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

```

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

```

(End of definition for __tag_struct_prop_gput:nnn.)

4.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object is @@/struct/1 which is currently created in the tree code (TODO move it here). The ParentTree and RoleMap entries are added at begin document in the tree code as they refer to object which are setup in other parts of the code. This avoid timing issues.

```

96 \<*package>
97 \tl_gset:Nn \g__tag_struct_stack_current_tl {1}

```

__tag_pdf_name_e:n

```

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

```

(End of definition for __tag_pdf_name_e:n.)

g__tag_struct_1_prop
g__tag_struct_kids_1_seq

```

100 \<*package>
101 \__tag_prop_new:c { g__tag_struct_1_prop }
102 \__tag_new_output_prop_handler:n {1}
103 \__tag_seq_new:c { g__tag_struct_kids_1_seq }
104
105 \__tag_struct_prop_gput:nne
106 { 1 }
107 { Type }
108 { \pdf_name_from_unicode_e:n {StructTreeRoot} }

```

```

109
110 \__tag_struct_prop_gput:nne
111   { 1 }
112   { S }
113   { \pdf_name_from_unicode_e:n {StructTreeRoot} }
114
115 \__tag_struct_prop_gput:nne
116   { 1 }
117   { rolemap }
118   { {StructTreeRoot}{pdf} }
119
120 \__tag_struct_prop_gput:nne
121   { 1 }
122   { parentrole }
123   { {StructTreeRoot}{pdf} }
124

```

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

```

125 \pdf_version_compare:NnF < {2.0}
126   {
127     \__tag_struct_prop_gput:nne
128       { 1 }
129       { Namespaces }
130       { \pdf_object_ref:n { __tag/tree/namespaces } }
131   }
132 </package>

```

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

```

133 <debug>\prop_new:c { g__tag_struct_debug_1_prop }
134 <debug>\seq_new:c { g__tag_struct_debug_kids_1_seq }
135 <debug>\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
136 <debug>\prop_gremove:cn { g__tag_struct_debug_1_prop }{Namespaces}

```

(End of definition for g__tag_struct_1_prop and g__tag_struct_kids_1_seq.)

4.2 Adding the /ID key

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

```

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

```

(End of definition for __tag_struct_get_id:n.)

4.3 Filling in the tag info

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

```

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

```

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

```

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

4.4 Handlings kids

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

`_tag_struct_kid_mc_gput_right:nn`
`_tag_struct_kid_mc_gput_right:ne`

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

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

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

`_tag_struct_kid_struct_gput_right:nn`
`_tag_struct_kid_struct_gput_right:ee`

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

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

```

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

```

(End of definition for __tag_struct_kid_struct_gput_right:nn.)

_tag_struct_kid_OBJR_gput_right:nn
_tag_struct_kid_OBJR_gput_right:eee

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

```

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

```

(End of definition for __tag_struct_kid_OBJR_gput_right:nnn.)

_tag_struct_exchange_kid_command:N
_tag_struct_exchange_kid_command:c

In luamode it can happen that a single kid in a structure is split at a page break into two or more mcid. In this case the lua code has to convert put the dictionary of the kid into an array. See issue 13 at tagpdf repo. We exchange the dummy command for the kids to mark this case. Change 2024-03-19: don't use a regex - that is slow.

```

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

```

```

267 \seq_gpop_left:NN #1 \l__tag_tmpa_tl
268 \tl_replace_once:Nnn \l__tag_tmpa_tl
269   {\__tag_mc_insert_mcid_kids:n}
270   {\__tag_mc_insert_mcid_single_kids:n}
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:NTF\g__tag_mode_lua_bool
304         {
305           \__tag_struct_exchange_kid_command:c
306           {g__tag_struct_kids_#1_seq}

```

check if we get null

```

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

```



```

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

```

(End of definition for __tag_struct_fill_kid_key:n.)

4.5 Output of the object

__tag_struct_get_dict_content:nM

This maps the dictionary content of a structure into a tl-var. Basically it does what \pdfdict_use:n does. This is used a lot so should be rather fast.

```

355 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
356 {
357     \tl_clear:N #2
358     \prop_map_inline:cn { g__tag_struct_#1_prop }
359     {
360         \tl_put_right:Ne #2

```

```
361         {
```

Some keys needs the option to format the value, e.g. add brackets for an array, we also need the option to ignore some entries in the properties.

```
362         \cs_if_exist_use:cTF {__tag_struct_format_##1:nn}
363         {##1}{##2}}
364         {\c_space_tl/##1~##2}
365     }
366 }
367 }
```

(End of definition for `__tag_struct_get_dict_content:nN`.)

```
\__tag_struct_format_rolemap:nn
\__tag_struct_format_parentrole:nn
```

This two entries should not end in the PDF.

```
368 \cs_new:Nn\__tag_struct_format_rolemap:nn{}
369 \cs_new:Nn\__tag_struct_format_parentrole:nn{}
```

(End of definition for `__tag_struct_format_rolemap:nn` and `__tag_struct_format_parentrole:nn`.)

```
\__tag_struct_format_Ref:nn
```

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

```
370 \cs_new:Nn\__tag_struct_format_Ref:nn{\c_space_tl/#1~##2}}
```

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

```
\__tag_struct_write_obj:n
```

This writes out the structure object. This is done in the finish code, in the tree module and guarded by the tree boolean.

```
371 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
372 {
373     \prop_if_exist:cTF { g__tag_struct_#1_prop }
374     {
```

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.

```
375     \prop_get:cnNF { g__tag_struct_#1_prop } {P}\l__tag_tmpb_tl
376     {
377         \prop_gput:cne { g__tag_struct_#1_prop } {P}{\pdf_object_ref_indexed:nn { __tag/
378         \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
379         \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
380         {
381             \msg_warning:nnee
382             {tag}
383             {struct-orphan}
384             { #1 }
385             {\seq_count:c{g__tag_struct_kids_#1_seq}}
386         }
387     }
388     \__tag_struct_fill_kid_key:n { #1 }
389     \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
390     \pdf_object_write_indexed:nne
391     { __tag/struct }{ #1 }
392     {dict}
393     {
394         \l__tag_tmpa_tl\c_space_tl
395         /ID-\__tag_struct_get_id:n{#1}
```

```

396         }
397     }
398 }
399 {
400     \msg_error:nnn { tag } { struct-no-objnum } { #1}
401 }
402 }

```

(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:

```

```

403 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat
404                                     %#2 structparent number
405 {
406     \bool_if:NT \g__tag_active_struct_bool
407     {
408         %get the number of the parent structure:
409         \seq_get:NNF
410             \g__tag_struct_stack_seq
411             \l__tag_struct_stack_parent_tmpa_tl
412         {
413             \msg_error:nn { tag } { struct-faulty-nesting }
414         }
415         %put the obj number of the annot in the kid entry, this also creates
416         %the OBJR object
417         \__tag_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
418         \__tag_struct_kid_OBJR_gput_right:eee
419         {
420             \l__tag_struct_stack_parent_tmpa_tl
421         }
422         {
423             #1 %
424         }

```

```

425     {
426     \pdf_pageobject_ref:n { \__tag_property_ref:nnn {@tag@objr@page@#2 }{ tagabspage
427     }
428     % add the parent obj number to the parent tree:
429     \exp_args:Nne
430     \__tag_parenttree_add_objr:nn
431     {
432     #2
433     }
434     {
435     \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl
436     }
437     % increase the int:
438     \int_gincr:N \c@g__tag_parenttree_obj_int
439     }
440 }

```

(End of definition for __tag_struct_insert_annot:mn.)

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

```

441 \cs_new:Npn \__tag_get_data_struct_tag:
442 {
443     \exp_args:Ne
444     \tl_tail:n
445     {
446     \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
447     }
448 }

```

(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**.

```

449 \cs_new:Npn \__tag_get_data_struct_id:
450 {
451     \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
452 }
453 </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

```

454 <*base>
455 \cs_new:Npn \__tag_get_data_struct_num:
456 {
457     \g__tag_struct_stack_current_tl
458 }
459 </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.

```

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

```

(End of definition for `_tag_get_data_struct_counter:.`)

5 Keys

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

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

```

466 <*package>
467 \socket_new:nn { tag/struct/tag }{1}
468 \socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
469 {
470   \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
471   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
472   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
473   \_tag_check_structure_tag:N \g__tag_struct_tag_tl
474 }
475
476 \socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
477 {
478   \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
479   \tl_gset:Ne \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
480   \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
481   \_tag_role_get:VVNN \g__tag_struct_tag_tl\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl\l__tag_t
482   \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
483   \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
484   \_tag_check_structure_tag:N \g__tag_struct_tag_tl
485 }
486 \socket_assign_plug:nn { tag/struct/tag } {latex-tags}

```

```

label_(struct-key)
stash_(struct-key)
parent_(struct-key)
tag_(struct-key)
title_(struct-key)
title-o_(struct-key)
alt_(struct-key)
actualtext_(struct-key)
lang_(struct-key)
ref_(struct-key)
E_(struct-key)
487 \keys_define:nn { __tag / struct }
488 {
489   label .tl_set:N      = \l__tag_struct_key_label_tl,
490   stash .bool_set:N   = \l__tag_struct_elem_stash_bool,
491   parent .code:n      =
492     {
493       \bool_lazy_and:nnTF
494         {
495           \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
496         }
497     }

```

```

498         \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
499     }
500     { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
501     {
502         \msg_warning:nnee { tag } { struct-unknown }
503         { \int_eval:n {#1} }
504         { parent~key~ignored }
505     }
506 },
507 parent .default:n      = {-1},
508 tag .code:n           = % S property
509     {
510         \socket_use:nn { tag/struct/tag }{#1}
511     },
512 title .code:n         = % T property
513     {
514         \str_set_convert:Nnnn
515         \l__tag_tmpa_str
516         { #1 }
517         { default }
518         { utf16/hex }
519         \__tag_struct_prop_gput:nne
520         { \int_use:N \c@g__tag_struct_abs_int }
521         { T }
522         { <\l__tag_tmpa_str> }
523     },
524 title-o .code:n       = % T property
525     {
526         \str_set_convert:Nonn
527         \l__tag_tmpa_str
528         { #1 }
529         { default }
530         { utf16/hex }
531         \__tag_struct_prop_gput:nne
532         { \int_use:N \c@g__tag_struct_abs_int }
533         { T }
534         { <\l__tag_tmpa_str> }
535     },
536 alt .code:n          = % Alt property
537     {
538         \tl_if_empty:oF{#1}
539         {
540             \str_set_convert:Noon
541             \l__tag_tmpa_str
542             { #1 }
543             { default }
544             { utf16/hex }
545             \__tag_struct_prop_gput:nne
546             { \int_use:N \c@g__tag_struct_abs_int }
547             { Alt }
548             { <\l__tag_tmpa_str> }
549         }
550     },
551 alttext .meta:n     = {alt=#1},

```

```

552 actualtext .code:n = % ActualText property
553 {
554   \tl_if_empty:oF{#1}
555   {
556     \str_set_convert:Noon
557     \l__tag_tmpa_str
558     { #1 }
559     { default }
560     { utf16/hex }
561     \__tag_struct_prop_gput:nne
562     { \int_use:N \c@g__tag_struct_abs_int }
563     { ActualText }
564     { <\l__tag_tmpa_str>}
565   }
566 },
567 lang .code:n = % Lang property
568 {
569   \__tag_struct_prop_gput:nne
570   { \int_use:N \c@g__tag_struct_abs_int }
571   { Lang }
572   { (#1) }
573 },

```

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

```

574 ref .code:n = % ref property
575 {
576   \tl_clear:N\l__tag_tmpa_tl
577   \clist_map_inline:on {#1}
578   {
579     \tl_put_right:Ne \l__tag_tmpa_tl
580     {-\__tag_property_ref:en{tagpdfstruct-##1}{tagstructobj} }
581   }
582   \__tag_struct_gput_data_ref:ee
583   { \int_use:N \c@g__tag_struct_abs_int } {\l__tag_tmpa_tl}
584 },
585 E .code:n = % E property
586 {
587   \str_set_convert:Nnon
588   \l__tag_tmpa_str
589   { #1 }
590   { default }
591   { utf16/hex }
592   \__tag_struct_prop_gput:nne
593   { \int_use:N \c@g__tag_struct_abs_int }
594   { E }
595   { <\l__tag_tmpa_str> }
596 },
597 }

```

(End of definition for label (struct-key) and others. These functions are documented on page 101.)

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

AFref_□(struct-key)

AFinline_□(struct-key)

AFinline-o_□(struct-key)

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

```

598 \int_new:N\g__tag_struct_AFobj_int

599 \cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
\g__tag_struct_AFobj_int 600 \cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
601 % #1 content, #2 extension
602 {
603   \tl_if_empty:nF{#1}
604   {
605     \group_begin:
606     \int_gincr:N \g__tag_struct_AFobj_int
607     \pdffile_embed_stream:neN
608     {#1}
609     {tag-Afile\int_use:N\g__tag_struct_AFobj_int.#2}
610     \l__tag_tmpa_tl
611     \__tag_struct_add_AF:ee
612     { \int_use:N \c@g__tag_struct_abs_int }
613     { \l__tag_tmpa_tl }
614     \__tag_struct_prop_gput:nne
615     { \int_use:N \c@g__tag_struct_abs_int }
616     { AF }
617     {
618       [
619         \tl_use:c
620         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
621       ]
622     }
623     \group_end:
624   }
625 }
626
627 \cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
628 \cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2 % #1 struct num #2 object reference
629 {
630   \tl_if_exist:cTF
631   {
632     g__tag_struct_#1_AF_tl
633   }
634   {
635     \tl_gput_right:ce
636     { g__tag_struct_#1_AF_tl }
637     { \c_space_tl #2 }
638   }
639   {
640     \tl_new:c
641     { g__tag_struct_#1_AF_tl }
642     \tl_gset:ce
643     { g__tag_struct_#1_AF_tl }
644     { #2 }
645   }
646 }

```



```

647 \cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
648 \keys_define:nn { __tag / struct }
649 {
650   AF .code:n          = % AF property
651   {
652     \pdf_object_if_exist:eTF {#1}
653     {
654       \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:
655       \__tag_struct_prop_gput:nne
656       { \int_use:N \c@g__tag_struct_abs_int }
657       { AF }
658       {
659         [
660           \tl_use:c
661           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
662         ]
663       }
664     }
665     {
666       % message?
667     }
668   },
669   AFref .code:n       = % AF property
670   {
671     \tl_if_empty:eF {#1}
672     {
673       \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
674       \__tag_struct_prop_gput:nne
675       { \int_use:N \c@g__tag_struct_abs_int }
676       { AF }
677       {
678         [
679           \tl_use:c
680           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
681         ]
682       }
683     }
684   },
685   ,AFinline .code:n =
686   {
687     \__tag_struct_add_inline_AF:nn {#1}{txt}
688   }
689   ,AFinline-o .code:n =
690   {
691     \__tag_struct_add_inline_AF:on {#1}{txt}
692   }
693   ,texsource .code:n =
694   {
695     \group_begin:
696     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
697     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
698     \__tag_struct_add_inline_AF:on {#1}{tex}
699     \group_end:
700   }

```

```

701 ,mathml .code:n =
702   {
703     \group_begin:
704     \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
705     \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
706     \__tag_struct_add_inline_AF:on {#1}{xml}
707     \group_end:
708   }
709 }

```

(End of definition for AF (struct-key) and others. These functions are documented on page 102.)

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!

```

710 \keys_define:nn { __tag / setup }
711   {
712     root-AF .code:n =
713     {
714       \pdf_object_if_exist:nTF {#1}
715       {
716         \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
717         \__tag_struct_prop_gput:nne
718         { 1 }
719         { AF }
720         {
721           [
722             \tl_use:c
723             { g__tag_struct_1_AF_t1 }
724           ]
725         }
726       }
727     }
728   }
729 }
730 },
731 }

```

(End of definition for root-AF (setup-key). This function is documented on page 103.)

6 User commands

We allow to set a language by default

```
\l__tag_struct_lang_tl
```

```

732 \tl_new:N \l__tag_struct_lang_tl
733 </package>

```

(End of definition for `\l__tag_struct_lang_tl`.)

```

\tag_struct_begin:n
\tag_struct_end:
734 <base>\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
735 <base>\cs_new_protected:Npn \tag_struct_end: {}
736 <base>\cs_new_protected:Npn \tag_struct_end:n {}
737 <*package | debug>

```

```

738 <package>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
739 <debug>\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
740 {
741 <package>\__tag_check_if_active_struct:T
742 <debug>\__tag_check_if_active_struct:TF
743 {
744   \group_begin:
745   \int_gincr:N \c@g__tag_struct_abs_int
746   \__tag_prop_new:c { g__tag_struct \int_eval:n { \c@g__tag_struct_abs_int }_prop }
747 <debug>   \prop_new:c { g__tag_struct_debug \int_eval:n { \c@g__tag_struct_abs_int }_prop
748   \__tag_new_output_prop_handler:n { \int_eval:n { \c@g__tag_struct_abs_int } }
749   \__tag_seq_new:c { g__tag_struct_kids \int_eval:n { \c@g__tag_struct_abs_int }_seq }
750 <debug>   \seq_new:c { g__tag_struct_debug_kids \int_eval:n { \c@g__tag_struct_abs_int }_
751   \pdf_object_new_indexed:nn { __tag/struct }
752   { \c@g__tag_struct_abs_int }
753   \__tag_struct_prop_gput:nnn
754   { \int_use:N \c@g__tag_struct_abs_int }
755   { Type }
756   { /StructElem }
757   \tl_if_empty:NF \l__tag_struct_lang_tl
758   {
759     \__tag_struct_prop_gput:nne
760     { \int_use:N \c@g__tag_struct_abs_int }
761     { Lang }
762     { (\l__tag_struct_lang_tl) }
763   }
764   \__tag_struct_prop_gput:nnn
765   { \int_use:N \c@g__tag_struct_abs_int }
766   { Type }
767   { /StructElem }
768
769   \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
770   \keys_set:nn { __tag / struct } { #1 }
771
772   \__tag_struct_set_tag_info:eVV
773   { \int_use:N \c@g__tag_struct_abs_int }
774   \g__tag_struct_tag_tl
775   \g__tag_struct_tag_NS_tl
776   \__tag_check_structure_has_tag:n { \int_use:N \c@g__tag_struct_abs_int }
777   \tl_if_empty:NF
778   \l__tag_struct_key_label_tl
779   {
780     \__tag_property_record:eV
781     {tagpdfstruct-\l__tag_struct_key_label_tl}
782     \c__tag_property_struct_clist
783   }

```

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

```

783   \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
784   {
785     \seq_get:NNF
786     \g__tag_struct_stack_seq
787     \l__tag_struct_stack_parent_tmpa_tl
788     {

```

```

789         \msg_error:nn { tag } { struct-faulty-nesting }
790     }
791 }
792 \seq_gpush:NV \g__tag_struct_stack_seq          \c@g__tag_struct_abs_int
793 \__tag_role_get:VVNN
794   \g__tag_struct_tag_tl
795   \g__tag_struct_tag_NS_tl
796   \l__tag_struct_roletag_tl
797   \l__tag_struct_roletag_NS_tl

```

to target role and role NS

```

798   \__tag_struct_prop_gput:nne
799   { \int_use:N \c@g__tag_struct_abs_int }
800   { rolemap }
801   {
802     {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
803   }

```

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!

```

804   \str_case:VnTF \l__tag_struct_roletag_tl
805   {
806     {Part} {}
807     {Div} {}
808     {NonStruct} {}
809   }
810   {
811     \prop_get:cnNT
812     { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
813     { parentrole }
814     \l__tag_get_tmpc_tl
815     {
816       \__tag_struct_prop_gput:nno
817       { \int_use:N \c@g__tag_struct_abs_int }
818       { parentrole }
819       {
820         \l__tag_get_tmpc_tl
821       }
822     }
823   }
824   {
825     \__tag_struct_prop_gput:nne
826     { \int_use:N \c@g__tag_struct_abs_int }
827     { parentrole }
828     {
829       {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
830     }
831   }
832   \seq_gpush:Ne \g__tag_struct_tag_stack_seq
833   {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
834   \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
835   %\seq_show:N \g__tag_struct_stack_seq
836   \bool_if:NF

```

```

837     \l__tag_struct_elem_stash_bool
838     {

```

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!

```

839     \__tag_struct_get_parentrole:eNN
840     {\l__tag_struct_stack_parent_tpa_tl}
841     \l__tag_get_parent_tpa_tl
842     \l__tag_get_parent_tpb_tl
843     \__tag_check_parent_child:VVVVN
844     \l__tag_get_parent_tpa_tl
845     \l__tag_get_parent_tpb_tl
846     \g__tag_struct_tag_tl
847     \g__tag_struct_tag_NS_tl
848     \l__tag_parent_child_check_tl
849     \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
850     {
851         \prop_get:cnN
852         { g__tag_struct_ \l__tag_struct_stack_parent_tpa_tl _prop}
853         {S}
854         \l__tag_tpa_tl
855         \quark_if_no_value:NT\l__tag_tpa_tl{\tl_set:Nn \l__tag_tpa_tl{UNKNOWN}}
856         \msg_warning:nneee
857         { tag }
858         {role-parent-child}
859         { \l__tag_get_parent_tpa_tl/\l__tag_get_parent_tpb_tl }
860         { \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
861         { not~allowed~
862           (struct~\l__tag_struct_stack_parent_tpa_tl,~\l__tag_tpa_tl
863            \c_space_tl-->~struct~\int_eval:n {\c@g__tag_struct_abs_int})
864         }
865         \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
866         \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
867         \__tag_role_remap:
868         \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
869         \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
870         \__tag_struct_set_tag_info:eVV
871         { \int_use:N \c@g__tag_struct_abs_int }
872         \g__tag_struct_tag_tl
873         \g__tag_struct_tag_NS_tl
874     }

```

Set the Parent.

```

875     \__tag_struct_prop_gput:nne
876     { \int_use:N \c@g__tag_struct_abs_int }
877     { P }
878     {
879         \pdf_object_ref_indexed:nn { __tag/struct} { \l__tag_struct_stack_parent_tpa_tl }
880     }
881     %record this structure as kid:
882     \%tl_show:N \g__tag_struct_stack_current_tl
883     \%tl_show:N \l__tag_struct_stack_parent_tpa_tl
884     \__tag_struct_kid_struct_gput_right:ee

```

```

885         { \l__tag_struct_stack_parent_tmpa_tl }
886         { \g__tag_struct_stack_current_tl }
887         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
888         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
889     }

```

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.

```

890 <debug>         \prop_gset_eq:cc
891 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
892 <debug>         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
893 <debug>         \prop_gput:cne
894 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
895 <debug>         { P }
896 <debug>         {
897 <debug>             \bool_if:NTF \l__tag_struct_elem_stash_bool
898 <debug>             {no~parent:~stashed}
899 <debug>             {
900 <debug>                 parent~structure:~\l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
901 <debug>                 \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_tl _p
902 <debug>             }
903 <debug>         }
904 <debug>         \prop_gput:cne
905 <debug>         { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
906 <debug>         { NS }
907 <debug>         { \g__tag_struct_tag_NS_tl }
908         %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
909         %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
910 <debug> \__tag_debug_struct_begin_insert:n { #1 }
911         \group_end:
912     }
913 <debug>{ \__tag_debug_struct_begin_ignore:n { #1 }}
914 }
915 <package>\cs_set_protected:Nn \tag_struct_end:
916 <debug>\cs_set_protected:Nn \tag_struct_end:
917 { %take the current structure num from the stack:
918   %the objects are written later, lua mode hasn't all needed info yet
919   %\seq_show:N \g__tag_struct_stack_seq
920 <package>\__tag_check_if_active_struct:T
921 <debug>\__tag_check_if_active_struct:TF
922 {
923     \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
924     \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
925     {
926         \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
927     }
928     { \__tag_check_no_open_struct: }
929     % get the previous one, shouldn't be empty as the root should be there
930     \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
931     {
932         \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
933     }
934     {

```

```

935     \__tag_check_no_open_struct:
936   }
937   \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
938   {
939     \tl_gset:Ne \g__tag_struct_tag_tl
940     { \exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl }
941     \prop_get:NVNT\g__tag_role_tags_NS_prop \g__tag_struct_tag_tl\l__tag_tmpa_tl
942     {
943       \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
944     }
945   }
946   <debug>\__tag_debug_struct_end_insert:
947   }
948   <debug>{\__tag_debug_struct_end_ignore:}
949   }
950
951   \cs_set_protected:Npn \tag_struct_end:n #1
952   {
953     <debug> \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
954     \tag_struct_end:
955   }
956   </package | debug>

```

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 100.)

\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.

```

957   <base>\cs_new_protected:Npn \tag_struct_use:n #1 {}
958   <*package | debug>
959   \cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
960   {
961     \__tag_check_if_active_struct:T
962     {
963       \prop_if_exist:cTF
964       { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop }
965       {
966         \__tag_check_struct_used:n {#1}
967         %add the label structure as kid to the current structure (can be the root)
968         \__tag_struct_kid_struct_gput_right:ee
969         { \g__tag_struct_stack_current_tl }
970         { \__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }
971         %add the current structure to the labeled one as parents
972         \__tag_prop_gput:cne
973         { g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{1}_prop }
974         { P }
975         {
976           \pdf_object_ref_indexed:nn { __tag/struct } { \g__tag_struct_stack_current_tl }
977         }
978       }
979     }

```

debug code

```

978   <debug> \prop_gput:cne
979   <debug> { g__tag_struct_debug_\__tag_property_ref:enn{tagpdfstruct-
980   <debug> { P }

```

```

981 <debug>          {
982 <debug>          parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
983 <debug>          \g__tag_struct_tag_tl
984 <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:

```

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

```

(End of definition for \tag_struct_use:n. This function is documented on page 100.)

\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.

```

1014 <base>\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
1015 <*package | debug>
1016 \cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
1017 {
1018   \__tag_check_if_active_struct:T
1019   {
1020     \prop_if_exist:cTF
1021     { g__tag_struct_#1_prop } %
1022     {
1023       \prop_get:cnNT
1024       {g__tag_struct_#1_prop}
1025       {P}

```



```

1026     \l__tag_tmpa_tl
1027     {
1028         \msg_warning:nnn { tag } {struct-used-twice} {#1}
1029     }
1030 %add the \#1 structure as kid to the current structure (can be the root)
1031 \__tag_struct_kid_struct_gput_right:ee
1032     { \g__tag_struct_stack_current_tl }
1033     { #1 }
1034 %add the current structure to \#1 as parent
1035 \__tag_struct_prop_gput:nne
1036     { #1 }
1037     { P }
1038     {
1039         \pdf_object_ref_indexed:nn { __tag/struct }{ \g__tag_struct_stack_current_tl
1040     }
1041 <debug>     \prop_gput:cne
1042 <debug>         { \g__tag_struct_debug_#1_prop }
1043 <debug>         { P }
1044 <debug>         {
1045 <debug>             parent~structure:~\g__tag_struct_stack_current_tl\c_space_tl=~
1046 <debug>             \g__tag_struct_tag_tl
1047 <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:

```

1048     \__tag_struct_get_parentrole:eNN
1049     {#1}
1050     \l__tag_tmpa_tl
1051     \l__tag_tmpb_tl
1052 \__tag_check_parent_child:VVVVN
1053     \g__tag_struct_tag_tl
1054     \g__tag_struct_tag_NS_tl
1055     \l__tag_tmpa_tl
1056     \l__tag_tmpb_tl
1057     \l__tag_parent_child_check_tl
1058 \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
1059     {
1060         \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
1061         \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
1062         \__tag_role_remap:
1063         \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
1064         \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
1065         \__tag_struct_set_tag_info:eVV
1066         { \int_use:N \c@g__tag_struct_abs_int }
1067         \g__tag_struct_tag_tl
1068         \g__tag_struct_tag_NS_tl
1069     }
1070 }
1071 {
1072     \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
1073 }
1074 }
1075 }
1076 </package | debug>

```

(End of definition for `\tag_struct_use_num:n`. This function is documented on page 100.)

`\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.

```

1077 <*package>
1078 \cs_new:Npn \tag_struct_object_ref:n #1
1079 {
1080   \pdf_object_ref_indexed:nn {__tag/struct}{#1 }
1081 }
1082 \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 100.)

`\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)

```

1083 \cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3
1084 {
1085   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
1086   { %warning??
1087     \use_none:nn
1088   }
1089   {#1}{#3}
1090 }
1091 \cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
1092 </package>

```

(End of definition for `\tag_struct_gput:nnn`. This function is documented on page 101.)

`__tag_struct_gput_data_ref:nn`

```

1093 <*package>
1094 \cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
1095   % #1 receiving struct num, #2 list of object ref
1096   {
1097     \prop_get:cnN
1098     { g__tag_struct_#1_prop }
1099     { Ref }
1100     \l__tag_get_tmpc_tl
1101     \__tag_struct_prop_gput:nne
1102     { #1 }
1103     { Ref }
1104     { \quark_if_no_value:Nf\l__tag_get_tmpc_tl { \l__tag_get_tmpc_tl\c_space_tl }#2 }
1105   }
1106 \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:` These 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.

```

1107 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
1108                                     %#2 struct parent num
1109 {
1110   \__tag_check_if_active_struct:T
1111   {
1112     \__tag_struct_insert_annot:nn {#1}{#2}
1113   }
1114 }
1115
1116 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx,ee}
1117 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
1118
1119 \end{package}
1120

```

(End of definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:`. These functions are documented on page 100.)

7 Attributes and attribute classes

```

1121 \begin{header}
1122 \ProvidesExplPackage {tagpdf-attr-code} {2024-06-20} {0.99c}
1123   {part of tagpdf - code related to attributes and attribute classes}
1124 \end{header}

```

7.1 Variables

`\g__tag_attr_entries_prop` will store attribute names and their dictionary content.
`\g__tag_attr_class_used_prop` will hold the attributes which have been used as class name. `\l__tag_attr_value_tl` is used to build the attribute array or key. Every time an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in `\g__tag_attr_objref_prop`

```

1125 \begin{package}
1126 \prop_new:N \g__tag_attr_entries_prop
1127 \prop_new_linked:N \g__tag_attr_class_used_prop
1128 \tl_new:N \l__tag_attr_value_tl
1129 \prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes

```

This seq is currently kept for compatibility with the table code.

```

1130 \seq_new:N \g__tag_attr_class_used_seq

```

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

7.2 Commands and keys

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

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

```

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

1131 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
1132 {
1133   \prop_gput:Nen \g__tag_attr_entries_prop
1134   {\pdf_name_from_unicode_e:n{#1}}{#2}
1135 }
1136
1137 \cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
1138 \keys_define:nn { __tag / setup }
1139 {
1140   role/new-attribute .code:n =
1141   {
1142     \__tag_attr_new_entry:nn #1
1143   }
}

deprecated name
1144   ,newattribute .code:n =
1145   {
1146     \__tag_attr_new_entry:nn #1
1147   },
1148 }

```

(End of definition for `__tag_attr_new_entry:nn`, `role/new-attribute` (setup-key), and `newattribute` (deprecated). These functions are documented on page 103.)

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

```

1149 \keys_define:nn { __tag / struct }
1150 {
1151   attribute-class .code:n =
1152   {
1153     \clist_set:Ne \l__tag_tmpa_clist { #1 }
1154     \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
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     \seq_map_inline:Nn \l__tag_tmpa_seq
1160     {
1161       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1162       {
1163         \msg_error:nnn { tag } { attr-unknown } { ##1 }
1164       }
1165       \prop_gput:Nnn \g__tag_attr_class_used_prop { ##1 } {}
1166     }
1167     \tl_set:Ne \l__tag_tmpa_tl

```

```

1168     {
1169         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1170         \seq_use:Nn \l__tag_tmpa_seq { \c_space_tl }
1171         \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
1172     }
1173     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
1174     {
1175         \__tag_struct_prop_gput:nne
1176         { \int_use:N \c@g__tag_struct_abs_int }
1177         { C }
1178         { \l__tag_tmpa_tl }
1179         %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
1180     }
1181 }
1182 }

```

(End of definition for attribute-class (struct-key). This function is documented on page 103.)

attribute_(struct-key)

```

1183 \keys_define:nn { __tag / struct }
1184 {
1185     attribute .code:n = % A property (attribute, value currently a dictionary)
1186     {
1187         \clist_set:Ne          \l__tag_tmpa_clist { #1 }
1188         \clist_if_empty:NF \l__tag_tmpa_clist
1189         {
1190             \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
1191             we convert the names into pdf names with slash
1192             \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
1193             {
1194                 \pdf_name_from_unicode_e:n {##1}
1195             }
1196             \tl_set:Ne \l__tag_attr_value_tl
1197             {
1198                 \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]%}
1199             }
1200             \seq_map_inline:Nn \l__tag_tmpa_seq
1201             {
1202                 \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
1203                 {
1204                     \msg_error:nnn { tag } { attr-unknown } { ##1 }
1205                 }
1206                 \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
1207                 {%\prop_show:N \g__tag_attr_entries_prop
1208                 \pdf_object_unnamed_write:ne
1209                 { dict }
1210                 {
1211                     \prop_item:Nn\g__tag_attr_entries_prop {##1}
1212                 }
1213                 \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
1214             }
1215             \tl_put_right:Ne \l__tag_attr_value_tl
1216             {
1217                 \c_space_tl

```

```

1217         \prop_item:Nn \g__tag_attr_objref_prop {##1}
1218     }
1219 % \tl_show:N \l__tag_attr_value_tl
1220 }
1221 \tl_put_right:Ne \l__tag_attr_value_tl
1222 { %[
1223     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
1224 }
1225 % \tl_show:N \l__tag_attr_value_tl
1226 \__tag_struct_prop_gput:mne
1227 { \int_use:N \c@g__tag_struct_abs_int }
1228 { A }
1229 { \l__tag_attr_value_tl }
1230 }
1231 },
1232 }
1233 \end{package}

```

(End of definition for attribute (struct-key). This function is documented on page 103.)

Part VIII

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2024-06-20} {0.99c}
4   {tagpdf-driver-for-luatex}
```

1 Loading the lua

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

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

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

```

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

```

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

```

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

```
68 </luatex>
```

The module declaration

```

69 <lua>
70 -- tagpdf.lua
71 -- Ulrike Fischer
72
73 local ProvidesLuaModule = {
74   name      = "tagpdf",
75   version   = "0.99c",      --TAGVERSION
76   date      = "2024-06-20", --TAGDATE
77   description = "tagpdf lua code",
78   license   = "The LATEX Project Public License 1.3c"
79 }

```



```

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

```

Some comments about the lua structure.

```

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

```

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

The `interwordspace attr` is set by the function `@@_mark_spaces`, and marks the place where spaces should be inserted. The `interwordfont attr` is set by the function `@@_mark_spaces` too and stores the font, so that we can decide which font to use for the real space char. The `interwordspaceOff attr` allows to locally suppress the insertion of real space chars, e.g. when they are inserted by other means (e.g. with `\char`).

```

134 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
135 local mcntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
136 local iwspaceOffattributeid = luatexbase.new_attribute ("g__tag_interwordspaceOff_attr")
137 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
138 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

```

139 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
140 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.

```

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

```

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

```

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

```

2 Logging functions

`__tag_log` This rather simple log function takes as argument a message (string) and a number and will output the message to the log/terminal if the current loglevel is greater or equal than num.

```

184 local __tag_log =
185   function (message,loglevel)
186     if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
187       texio.write_nl("tagpdf: ".. message)
188     end
189   end
190
191 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.

```

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

```

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

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

```

201 local __tag_pairs_prop =
202   function (prop)
203     local a = {}
204     for n in pairs(prop) do tableinsert(a, n) end
205     table.sort(a)

```

```

206     local i = 0                -- iterator variable
207     local iter = function ()  -- iterator function
208         i = i + 1
209         if a[i] == nil then return nil
210         else return a[i], prop[a[i]]
211         end
212     end
213     return iter
214 end
215
216
217 function ltx.__tag.trace.show_prop (prop)
218     if (type(prop) == "table") then
219         for i,v in __tag_pairs_prop (prop) do
220             __tag_log ("[" .. i .. "] => " .. tostring(v),1)
221         end
222     else
223         __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
224     end
225 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

```

226 function ltx.__tag.trace.show_mc_data (num,loglevel)
227     if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
228         for k,v in pairs(ltx.__tag.mc[num]) do
229             __tag_log ("mc"..num..": "..tostring(k).."=>"..tostring(v),loglevel)
230         end
231         if ltx.__tag.mc[num]["kids"] then
232             __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
233             for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
234                 __tag_log ("mc " .. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
235             end
236         end
237     else
238         __tag_log ("mc"..num.." not found",loglevel)
239     end
240 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.

```

241 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
242     for i = min, max do
243         ltx.__tag.trace.show_mc_data (i,loglevel)
244     end
245     texio.write_nl("")
246 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.

```
247 function ltx.__tag.trace.show_struct_data (num)
248   if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
249     for k,v in ipairs(ltx.__tag.struct[num]) do
250       __tag_log ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
251     end
252   else
253     __tag_log ("struct "..num.." not found ",1)
254   end
255 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).

```
256 local __tag_get_mc_cnt_type_tag = function (n)
257   local mcnt      = nodegetattribute(n,mcntattributeid) or -1
258   local mctype    = nodegetattribute(n,mctypeattributeid) or -1
259   local tag       = ltx.__tag.func.get_tag_from(mctype)
260   return mcnt,mctype,tag
261 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.

```
262 local function __tag_get_mathsubtype (mathnode)
263   if mathnode.subtype == 0 then
264     subtype = "beginmath"
265   else
266     subtype = "endmath"
267   end
268   return subtype
269 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.

```
270 ltx.__tag.tables.role_tag_attribute = {}
271 ltx.__tag.tables.role_attribute_tag = {}
```

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

ltx.__tag.func.alloctag

```
272 local __tag_alloctag =
273   function (tag)
274     if not ltx.__tag.tables.role_tag_attribute[tag] then
275       table.insert(ltx.__tag.tables.role_attribute_tag,tag)
```

```

276     ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
277     __tag_log ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
278     end
279 end
280 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.

```

281 local __tag_get_num_from =
282 function (tag)
283   if ltx.__tag.tables.role_tag_attribute[tag] then
284     a= ltx.__tag.tables.role_tag_attribute[tag]
285   else
286     a= -1
287   end
288   return a
289 end
290
291 ltx.__tag.func.get_num_from = __tag_get_num_from
292
293 function ltx.__tag.func.output_num_from (tag)
294   local num = __tag_get_num_from (tag)
295   tex.sprint(catlatex,num)
296   if num == -1 then
297     __tag_log ("Unknown tag "..tag.." used")
298   end
299 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.

```

300 local __tag_get_tag_from =
301 function (num)
302   if ltx.__tag.tables.role_attribute_tag[num] then
303     a = ltx.__tag.tables.role_attribute_tag[num]
304   else
305     a= "UNKNOWN"
306   end
307   return a
308 end
309
310 ltx.__tag.func.get_tag_from = __tag_get_tag_from
311
312 function ltx.__tag.func.output_tag_from (num)
313   tex.sprint(catlatex,__tag_get_tag_from (num))
314 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.

```

315 function ltx.__tag.func.store_mc_data (num,key,data)
316   ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
317   ltx.__tag.mc[num][key] = data
318   __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3)
319 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.

```

320 function ltx.__tag.func.store_mc_label (label,num)
321   ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
322   ltx.__tag.mc.labels[label] = num
323 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.

```

324 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
325   ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
326   ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
327   local kidtable = {kid=kid,page=page}
328   tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
329 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.

```

330 function ltx.__tag.func.mc_num_of_kids (mcnum)
331   local num = 0
332   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
333     num = #ltx.__tag.mc[mcnum]["kids"]
334   end
335   ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
336   return num
337 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

```

338 local __tag_backend_create_emc_node
339 if tex.outputmode == 0 then
340   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
341     function __tag_backend_create_emc_node ()
342       local emcnode = nodenew("whatsit","special")
343       emcnode.data = "pdf:code EMC"
344       return emcnode
345     end

```

```

346 else -- assume a dvips variant
347   function __tag_backend_create_emc_node ()
348     local emcnode = nodenew("whatsit","special")
349     emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
350     return emcnode
351   end
352 end
353 else -- pdf mode
354   function __tag_backend_create_emc_node ()
355     local emcnode = nodenew("whatsit","pdf_literal")
356     emcnode.data = "EMC"
357     emcnode.mode=1
358     return emcnode
359   end
360 end
361
362 local function __tag_insert_emc_node (head,current)
363   local emcnode= __tag_backend_create_emc_node()
364   head = node.insert_before(head,current,emcnode)
365   return head
366 end

```

(End of definition for __tag_backend_create_emc_node and __tag_insert_emc_node.)

__tag_backend_create_bmc_node
__tag_insert_bmc_node

This inserts a simple bmc node

```

367 local __tag_backend_create_bmc_node
368 if tex.outputmode == 0 then
369   if token.get_macro("c_sys_backend_str") == "dviptdpmx" then
370     function __tag_backend_create_bmc_node (tag)
371       local bmcnode = nodenew("whatsit","special")
372       bmcnode.data = "pdf:code /"..tag.." BMC"
373       return bmcnode
374     end
375   else -- assume a dvips variant
376     function __tag_backend_create_bmc_node (tag)
377       local bmcnode = nodenew("whatsit","special")
378       bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
379       return bmcnode
380     end
381   end
382 else -- pdf mode
383   function __tag_backend_create_bmc_node (tag)
384     local bmcnode = nodenew("whatsit","pdf_literal")
385     bmcnode.data = "/"..tag.." BMC"
386     bmcnode.mode=1
387     return bmcnode
388   end
389 end
390
391 local function __tag_insert_bmc_node (head,current,tag)
392   local bmcnode = __tag_backend_create_bmc_node (tag)
393   head = node.insert_before(head,current,bmcnode)
394   return head
395 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.

```
396 local __tag_backend_create_bdc_node
397
398 if tex.outputmode == 0 then
399   if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
400     function __tag_backend_create_bdc_node (tag,dict)
401       local bdcnode = nodenew("whatsit","special")
402       bdcnode.data = "pdf:code /"..tag.."<<..dict..>> BDC"
403       return bdcnode
404     end
405   else -- assume a dvips variant
406     function __tag_backend_create_bdc_node (tag,dict)
407       local bdcnode = nodenew("whatsit","special")
408       bdcnode.data = "ps:SDict begin mark/"..tag.."<<..dict..>> /BDC pdfmark end"
409       return bdcnode
410     end
411   end
412 else -- pdf mode
413   function __tag_backend_create_bdc_node (tag,dict)
414     local bdcnode = nodenew("whatsit","pdf_literal")
415     bdcnode.data = "/"..tag.."<<..dict..>> BDC"
416     bdcnode.mode=1
417     return bdcnode
418   end
419 end
420
421 local function __tag_insert_bdc_node (head,current,tag,dict)
422   bdcnode= __tag_backend_create_bdc_node (tag,dict)
423   head = node.insert_before(head,current,bdcnode)
424   return head
425 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.

```
426 local function __tag_pdf_object_ref (name,index)
427   local object
428   if ltx.pdf.object_id then
429     object = ltx.pdf.object_id (name,index) ..' 0 R'
430   else
431     local tokename = 'c__pdf_object_'..name..'/'..index..'_int'
432     object = token.create(tokename).mode ..' 0 R'
433   end
434   return object
435 end
436 ltx.__tag.func.pdf_object_ref = __tag_pdf_object_ref
```

(End of definition for `__tag_pdf_object_ref`.)

4 Function for the real space chars

`__tag_show_spacemark` A debugging function, it is used to inserts red color markers in the places where space chars can go, it can have side effects so not always reliable, but ok.

```
437 local function __tag_show_spacemark (head,current,color,height)
438   local markcolor = color or "1 0 0"
439   local markheight = height or 10
440   local pdfstring
441   if tex.outputmode == 0 then
442     -- ignore dvi mode for now
443   else
444     pdfstring = node.new("whatsit","pdf_literal")
445     pdfstring.data =
446       string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
447         3,markheight)
448     head = node.insert_after(head,current,pdfstring)
449   return head
450 end
```

(End of definition for `__tag_show_spacemark`.)

`__tag_fakespace` This is used to define a lua version of `\pdfspace`
`ltx.__tag.func.fakespace`

```
451 local function __tag_fakespace()
452   tex.setattribute(iwspaceattributeid,1)
453   tex.setattribute(iwfontattributeid,font.current())
454 end
455 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.

```
456 --[[ a function to mark up places where real space chars should be inserted
457     it only sets an attribute.
458 --]]
459
460 local function __tag_mark_spaces (head)
461   local inside_math = false
462   for n in nodetraverse(head) do
463     local id = n.id
464     if id == GLYPH then
465       local glyph = n
466       default_currfontid = glyph.font
467       if glyph.next and (glyph.next.id == GLUE)
468         and not inside_math and (glyph.next.width >0)
469       then
470         nodesetattribute(glyph.next,iwspaceattributeid,1)
471         nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
472       -- for debugging
473       if ltx.__tag.trace.showspace then
474         __tag_show_spacemark (head,glyph)
475       end
476     end
477   end
```

```

476 elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
477 local kern = glyph.next
478 if kern.next and (kern.next.id== GLUE) and (kern.next.width >0)
479 then
480 nodesetattribute(kern.next,iwspaceattributeid,1)
481 nodesetattribute(kern.next,iwfontattributeid,glyph.font)
482 end
483 end
484 -- look also back
485 if glyph.prev and (glyph.prev.id == GLUE)
486 and not inside_math
487 and (glyph.prev.width >0)
488 and not nodehasattribute(glyph.prev,iwspaceattributeid)
489 then
490 nodesetattribute(glyph.prev,iwspaceattributeid,1)
491 nodesetattribute(glyph.prev,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 == PENALTY then
498 local glyph = n
499 -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
500 if glyph.next and (glyph.next.id == GLUE)
501 and not inside_math and (glyph.next.width >0) and n.subtype==0
502 then
503 nodesetattribute(glyph.next,iwspaceattributeid,1)
504 -- changed 2024-01-18, issue #72
505 nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
506 -- for debugging
507 if ltx.__tag.trace.showspace then
508 __tag_show_spacemark (head,glyph)
509 end
510 end
511 elseif id == MATH then
512 inside_math = (n.subtype == 0)
513 end
514 end
515 return head
516 end

```

(End of definition for __tag_mark_spaces.)

```

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

```

These functions add/remove the function which marks the spaces to the callbacks `pre_linebreak_filter` and `hpack_filter`

```

517 local function __tag_activate_mark_space ()
518 if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
519 luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
520 luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
521 end
522 end
523
524 ltx.__tag.func.markspaceon=__tag_activate_mark_space

```

```

525
526 local function __tag_deactivate_mark_space ()
527   if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
528     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
529     luatexbase.remove_from_callback("hpack_filter","markspaces")
530   end
531 end
532
533 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.

```

534 local default_space_char = nodenew(GLYPH)
535 local default_fontid      = fontid("TU/lmr/m/n/10")
536 local default_currfontid = fontid("TU/lmr/m/n/10")
537 default_space_char.char   = 32
538 default_space_char.font   = default_fontid

```

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

```

539 local function __tag_font_has_space (fontid)
540   t= fonts.hashes.identifiers[fontid]
541   if luaotfload.aux.slot_of_name(fontid,"space")
542     or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
543   then
544     return true
545   else
546     return false
547   end
548 end

```

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

```

__tag_space_chars_shipout
ltx.__tag.func.space_chars_shipout

```

```

549 local function __tag_space_chars_shipout (box)
550   local head = box.head
551   if head then
552     for n in node.traverse(head) do
553       local spaceattr = -1
554       if not nodehasattribute(n,iwspaceoffattributeid) then
555         spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
556       end
557       if n.id == HLIST then -- enter the hlist
558         __tag_space_chars_shipout (n)
559       elseif n.id == VLIST then -- enter the vlist
560         __tag_space_chars_shipout (n)
561       elseif n.id == GLUE then
562         if ltx.__tag.trace.showspaces and spaceattr==1 then
563           __tag_show_spacemark (head,n,"0 1 0")
564         end
565         if spaceattr==1 then
566           local space
567           local space_char = node.copy(default_space_char)
568           local curfont     = nodegetattribute(n,iwfontattributeid)
569           ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)

```

```

570         if curfont and
571             -- luaotfload.aux.slot_of_name(curfont,"space")
572             __tag_font_has_space (curfont)
573         then
574             space_char.font=curfont
575         end
576         head, space = node.insert_before(head, n, space_char) --
577         n.width      = n.width - space.width
578         space.attr   = n.attr
579     end
580 end
581 end
582 box.head = head
583 end
584 end
585
586 function ltx.__tag.func.space_chars_shipout (box)
587     __tag_space_chars_shipout (box)
588 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.

```

589 function ltx.__tag.func.mc_insert_kids (mcnum, single)
590     if ltx.__tag.mc[mcnum] then
591         ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum, 4)
592         if ltx.__tag.mc[mcnum]["kids"] then
593             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
594                 tex.sprint("[")
595             end
596             for i, kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
597                 local kidnum = kidstable["kid"]
598                 local kidpage = kidstable["page"]
599                 local kidpageobjnum = pdfpageref(kidpage)
600                 ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
601                     " insert KID " .. i ..
602                     " with num " .. kidnum ..
603                     " on page " .. kidpage .. "/" .. kidpageobjnum, 3)
604                 tex.sprint(catlatex, "<</Type /MCR /Pg " .. kidpageobjnum .. " 0 R /MCID " .. kidnum .. ">> ")
605             end
606             if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
607                 tex.sprint("]")
608             end
609         else
610             -- this is typically not a problem, e.g. empty hbox in footer/header can
611             -- trigger this warning.
612             ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: " .. mcnum .. " has no kids", 2)
613             if single==1 then

```

```

614         tex.sprint("null")
615     end
616 end
617 else
618     ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
619 end
620 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.

```

621 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
622     ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
623     ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
624     -- a structure can contain more than on mc chunk, the content should be ordered
625     tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
626     ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
627         mcnum.." inserted in struct "..structnum,3)
628     -- but every mc can only be in one structure
629     ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
630     ltx.__tag.mc[mcnum]["parent"] = structnum
631 end
632

```

(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.

```

633 -- pay attention: lua counts arrays from 1, tex pages from one
634 -- mcid and arrays in pdf count from 0.
635 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagencnt,page)
636     ltx.__tag.page[page] = ltx.__tag.page[page] or {}
637     ltx.__tag.page[page][mcpagencnt] = mcnum
638     ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
639         ": inserting MCID " .. mcpagencnt .. " => " .. mcnum,3)
640 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)

```

641 local function __tag_update_mc_attributes (head,mcnum,type)
642     for n in node.traverse(head) do
643         node.set_attribute(n,mcntattributeid,mcnum)
644         node.set_attribute(n,mctypeattributeid,type)
645         if n.id == HLIST or n.id == VLIST then
646             __tag_update_mc_attributes (n.list,mcnum,type)
647         end
648     end
649     return head
650 end
651 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.

```
652 --[[
653     Now follows the core function
654     It wades through the shipout box and checks the attributes
655     ARGUMENTS
656     box: is a box,
657     mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed fo
658     mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a
659     mcopen: num, records if some bdc/emc is open
660     These arguments are only needed for log messages, if not present are replaces by fix str
661     name: string to describe the box
662     mctypeprev: num, the type attribute of the previous node/whatever
663
664     there are lots of logging messages currently. Should be cleaned up in due course.
665     One should also find ways to make the function shorter.
666 --]]
667
668 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
669     local name = name or ("SOMEBOX")
670     local mctypeprev = mctypeprev or -1
671     local abspage = status.total_pages + 1 -- the real counter is increased
672                                           -- inside the box so one off
673                                           -- if the callback is not used. (???)
674     ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
675     ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
676                         " prev "..mccntprev ..
677                         " type prev "..mctypeprev,4)
678     ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
679                         " TYPE ".. node.type(node.getid(box)),3)
680     local head = box.head -- ShipoutBox is a vlist?
681     if head then
682         mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
683         ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
684                             node.type(node.getid(head))..
685                             " MC"..tostring(mccnthead)..
686                             " => TAG " .. tostring(mctypehead)..
687                             " => ".. tostring(taghead),3)
688     else
689         ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
690                             tostring(head),3)
691     end
692     for n in node.traverse(head) do
693         local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
694         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
695         ltx.__tag.trace.log ("INFO TAG-NODE: "..
696                             node.type(node.getid(n))..
697                             " MC".. tostring(mccnt)..
698                             " => TAG ".. tostring(mctype)..
699                             " => " .. tostring(tag),3)
700         if n.id == HLIST
701         then -- enter the hlist
702             mcopen,mcpagecnt,mccntprev,mctypeprev=
```

```

703     ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcntprev,mcopen,"INTERNAL HLIST",mctype
704 elseif n.id == VLIST then -- enter the vlist
705     mcpopen,mcpagecnt,mcntprev,mctypeprev=
706     ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcntprev,mcopen,"INTERNAL VLIST",mctype
707 elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
708                                     -- been done if the previous shipout wandering, so here it
709 elseif n.id == LOCAL_PAR then -- local_par is ignored
710 elseif n.id == PENALTY then -- penalty is ignored
711 elseif n.id == KERN then -- kern is ignored
712     ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
713     node.type(node.getid(n)).." ".n.subtype,4)
714 else
715     -- math is currently only logged.
716     -- we could mark the whole as math
717     -- for inner processing the mlist_to_hlist callback is probably needed.
718     if n.id == MATH then
719         ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
720         node.type(node.getid(n)).." ".__tag_get_mathsubtype(n),4)
721     end
722     -- endmath
723     ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
724     mcnt.." prev "..mcntprev,4)
725     if mcnt~=mcntprev then -- a new mc chunk
726         ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
727         node.type(node.getid(n))..
728         " MC"..tostring(mcnt)..
729         " <=> PREVIOUS "..tostring(mcntprev),4)
730     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
731         box.list=__tag_insert_emc_node (box.list,n)
732         mcopen = mcopen - 1
733         ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
734         mcpagecnt .. " MCOPEX = " .. mcopen,3)
735         if mcopen ~=0 then
736             ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,1)
737         end
738     end
739     if ltx.__tag.mc[mcnt] then
740         if ltx.__tag.mc[mcnt]["artifact"] then
741             ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
742             tostring(ltx.__tag.mc[mcnt]["artifact"]),3)
743         if ltx.__tag.mc[mcnt]["artifact"] == "" then
744             box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
745         else
746             box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mc
747         end
748     else
749         ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
750         tostring(tag),3)
751         mcpagecnt = mcpagecnt +1
752         ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
753         local dict= "/MCID ".mcpagecnt
754         if ltx.__tag.mc[mcnt]["raw"] then
755             ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
756             tostring(ltx.__tag.mc[mcnt]["raw"]),3)

```



```

757         dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
758     end
759     if ltx.__tag.mc[mccnt]["alt"] then
760         ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
761             tostring(ltx.__tag.mc[mccnt]["alt"]),3)
762         dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
763     end
764     if ltx.__tag.mc[mccnt]["actualtext"] then
765         ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
766             tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
767         dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
768     end
769     box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
770     ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
771     ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
772     ltx.__tag.trace.show_mc_data (mccnt,3)
773 end
774 mcopen = mcopen + 1
775 else
776     if tagunmarkedbool.mode == truebool.mode then
777         ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
778         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
779         mcopen = mcopen + 1
780     else
781         ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
782     end
783 end
784 mccntprev = mccnt
785 end
786 end -- end if
787 end -- end for
788 if head then
789     mccnthead, mctypehead, taghead = __tag_get_mc_cnt_type_tag (head)
790     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
791         node.type(node.getid(head))..
792         " MC"..tostring(mccnthead)..
793         " => TAG "..tostring(mctypehead)..
794         " => "..tostring(taghead),4)
795 else
796     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
797 end
798 ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
799     tostring(name)..
800     " TYPE ".. node.type(node.getid(box)),4)
801 return mcopen,mcpagecnt,mccntprev,mctypeprev
802 end
803

```

(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.

```

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

```

```

805 mcopyen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
806 if mcopyen~=0 then -- there is a chunk open, close it (hope there is only one ...
807 local emcnode = __tag_backend_create_emc_node ()
808 local list = box.list
809 if list then
810     list = node.insert_after (list,node.tail(list),emcnode)
811     mcopyen = mcopyen - 1
812     ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPIEN " .. mcopyen,3)
813 else
814     ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
815 end
816 if mcopyen ~=0 then
817     ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopyen,1)
818 end
819 end
820 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.

```

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

```

```

851     local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
852     local proptime = "g__tag_struct_"..structnum.."_prop"
853     --local objref = ltx.__tag.tables[proptime]["objref"] or "XXXX"
854     local objref = __tag_pdf_object_ref('__tag/struct',structnum)
855     numentry = numentry .. " " .. objref
856     end
857     numentry = numentry .. "]" "
858     ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
859     page.. " num entry = ".. numentry,3)
860     end
861     else
862     ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
863     numentry = pdfpage.." ["
864     end
865     return numentry
866 end
867
868 function ltx.__tag.func.output_parenttree (abspage)
869 for i=1,abspage do
870 line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
871 tex.sprint(catlatex,line)
872 end
873 end

```

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

```

874 </lua>

```

Part IX

The `tagpdf-roles` module

Tags, roles and namespace code

Part of the `tagpdf` package

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

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

The key-value list knows the following keys:

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

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

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

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

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

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

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

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2024-06-20} {0.99c}
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 dependent props. With pdf <2.0 we store for now the type(s) of a tag in a common prop. Tags that are rolemapped should get the type from the target.

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

(End of definition for `\g__tag_role_tags_class_prop`.)

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

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

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

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

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

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

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

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

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

(End of definition for `\g__tag_role_NS_prop`.)

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

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

(End of definition for `\g__tag_role_index_prop`.)

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

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

(End of definition for `\l__tag_role_debug_prop`.)

We need also a bunch of temporary variables.

```
\l__tag_role_tag_tmpa_tl
```

```
\l__tag_role_tag_namespace_tmpa_tl
```

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

```
\l__tag_role_tag_namespace_tmpb_tl\l__tag_role_tag_namespace_tmpa_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_tmpa_seq
```

```
17 \seq_new:N \l__tag_role_tmpa_seq
```

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

1.2 Namespaces

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

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

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

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

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

```
\__tag_role_NS_new:nnn
```

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

```

52     {
53       \pdfdict_gput:nne{g__tag_role/namespace_#1_dict}
54       {Schema}{#3}
55     }
56     \prop_gput:Nne \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1~}}
57   }
58 }

```

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

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

`\c__tag_role_userNS_id_str`

```

59 \str_const:Ne \c__tag_role_userNS_id_str
60   { data:,
61     \int_to_Hex:n{\int_rand:n {65535}}
62     \int_to_Hex:n{\int_rand:n {65535}}
63     -
64     \int_to_Hex:n{\int_rand:n {65535}}
65     -
66     \int_to_Hex:n{\int_rand:n {65535}}
67     -
68     \int_to_Hex:n{\int_rand:n {65535}}
69     -
70     \int_to_Hex:n{\int_rand:n {16777215}}
71     \int_to_Hex:n{\int_rand:n {16777215}}
72   }

```

(End of definition for `\c__tag_role_userNS_id_str`.)

Now we setup the standard names spaces. The mathml space is loaded also for pdf < 2.0 but not added to RoleMap unless a boolean is set to true with `tagpdf-setup{mathml-tags}`.

```

73 \bool_new:N \g__tag_role_add_mathml_bool
74 \__tag_role_NS_new:nnn {pdf} {http://iso.org/pdf/ssn}{}
75 \__tag_role_NS_new:nnn {pdf2} {http://iso.org/pdf2/ssn}{}
76 \__tag_role_NS_new:nnn {mathml} {http://www.w3.org/1998/Math/MathML}{}
77 \__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dfl/2022}{}
78 \__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book/2022}{}
79 \exp_args:Nne
80   \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{}

```

1.3 Adding a new tag

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

`__tag_role_alloctag:nnn`

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

```

81 \pdf_version_compare:NnTF < {2.0}
82   {
83     \sys_if_engine luatex:TF
84     {

```



```

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

```

(End of definition for __tag_role_alloctag:nnn.)

1.3.1 pdf 1.7 and earlier

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

```

127 \cs_new_protected:Nn \__tag_role_add_tag:nn % (new) name, reference to old
128   {

```

checks and messages

```

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

```

now the addition

```

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

```

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

```

143 \tl_if_empty:nF { #2 }
144 {
145   \prop_get:NnN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
146   \quark_if_no_value:NTF \l__tag_tmpa_tl
147   {
148     \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
149   }
150   {
151     \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
152   }
153 }
154 }
155 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV,ne}

```

(End of definition for __tag_role_add_tag:nn.)

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

__tag_role_get:nnNN

```

156 \pdf_version_compare:NnT < {2.0}
157 {
158   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 %#1 tag, #2 NS, #3 tlvar which hold the role tag
159   {
160     \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
161     {
162       \tl_set:Nn #3 {#1}
163     }
164     \tl_set:Nn #4 {}
165   }
166   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
167 }
168

```

(End of definition for __tag_role_get:nnNN.)

1.3.2 The pdf 2.0 version

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

```

169 \cs_new_protected:Nn \_tag_role_add_tag:nnnn %tag/namespace/role/namespace
170 {
171   \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
172   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
173   {
174     \msg_info:nnn { tag }{new-tag}{#1}
175   }
176   \prop_if_exist:cTF
177   { g__tag_role_NS_#4_class_prop }
178   {
179     \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
180     \quark_if_no_value:NT \l__tag_tmpa_tl
181     {
182       \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
183     }
184   }
185   { \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--} }
186   \_tag_role_alloctag:nnV {#1}{#2}\l__tag_tmpa_tl
  
```

Do not remap standard tags. TODO add warning?

```

187 \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
188 {
189   \pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
190   {
191     [
192       \pdf_name_from_unicode_e:n{#3}
193       \c_space_tl
194       \pdf_object_ref:n {tag/NS/#4}
195     ]
196   }
197 }
  
```

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

```

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

We also store into the pdf 1.7 rolemapping so that we can add that as fallback for pdf 1.7 processor

```

210 \bool_if:NT \l__tag_role_update_bool
211 {
212   \tl_if_empty:nF { #3 }
  
```

```

213     {
214       \tl_if_eq:nnF{#1}{#3}
215       {
216         \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
217         \quark_if_no_value:NTF \l__tag_tmpa_tl
218         {
219           \prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
220         }
221         {
222           \prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
223         }
224       }
225     }
226   }
227 }
228 \cs_generate_variant:Nn \__tag_role_add_tag: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`

```

229 \pdf_version_compare:NnF < {2.0}
230 {
231   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
232     % #1 tag, #2 NS,
233     % #3 tlvar which hold the role tag
234     % #4 tlvar which hold the name of the target NS
235     {
236       \prop_if_exist:cTF {g__tag_role_NS_#2_prop}
237       {
238         \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
239         {
240           \tl_set:Ne #3 {\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
241           \tl_set:Ne #4 {\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
242         }
243         {
244           \msg_warning:nnn { tag } {role-unknown-tag} { #1 }
245           \tl_set:Nn #3 {#1}
246           \tl_set:Nn #4 {#2}
247         }
248       }
249       {
250         \msg_warning:nnn { tag } {role-unknown-NS} { #2 }
251         \tl_set:Nn #3 {#1}
252         \tl_set:Nn #4 {#2}
253       }
254     }
255   \cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
256 }

```

(End of definition for `__tag_role_get:nnNN`.)

1.4 Helper command to read the data from files

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

`_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.

```

257 \bool_new:N\l__tag_role_update_bool
258 \bool_set_true:N \l__tag_role_update_bool
259 \pdf_version_compare:NnTF < {2.0}
260 {
261   \cs_new_protected:Npn \_tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
262   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
263   {
264     \tl_if_empty:nF { #2 }
265     {
266       \bool_if:NTF \l__tag_role_update_bool
267       {
268         \tl_if_empty:nTF {#5}
269         {
270           \prop_get:NnN \g__tag_role_tags_class_prop {#3}\l__tag_tmpa_tl
271           \quark_if_no_value:NT \l__tag_tmpa_tl
272           {
273             \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
274           }
275         }
276         {
277           \tl_set:Nn \l__tag_tmpa_tl {#5}
278         }
279         \_tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
280         \tl_if_eq:nnF {#2}{#3}
281         {
282           \_tag_role_add_tag:nn {#2}{#3}
283         }
284         \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{#3}{}}
285       }
286       {
287         \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#{#3}{}}
288         \prop_gput:cnn {g__tag_role_NS_#1_class_prop} {#2}{--UNUSED--}
289       }
290     }
291   }
292 }
293 {
294   \cs_new_protected:Npn \_tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
295   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
296   {
297     \tl_if_empty:nF {#2}
298     {
299       \tl_if_empty:nTF {#5}
300       {
301         \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
302         \quark_if_no_value:NT \l__tag_tmpa_tl

```

```

303         {
304         \tl_set:Nn \l__tag_tmpa_tl {--UNKNOWN--}
305         }
306     }
307     {
308     \tl_set:Nn \l__tag_tmpa_tl {#5}
309     }
310     \__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
311     \bool_lazy_and:nnT
312     { ! \tl_if_empty_p:n {#3} } { ! \str_if_eq_p:nn {#1}{pdf2} }
313     {
314     \__tag_role_add_tag:nxxx {#2}{#1}{#3}{#4}
315     }
316     \prop_gput:cnn {g__tag_role_NS_#1_prop} {#2}{#3}{#4}
317 }
318 }
319 }

```

(End of definition for __tag_role_read_namespace_line:nw.)

__tag_role_read_namespace:nn

This command reads a namespace file in the format tagpdf-ns-XX.def

```

320 \cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
321 {
322     \prop_if_exist:cF {g__tag_role_NS_#1_prop}
323     { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
324     \file_if_exist:nTF { tagpdf-ns-#2.def }
325     {
326     \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
327     \msg_info:nnn {tag}{read-namespace}{#2}
328     \ior_map_inline:Nn \g_tmpa_ior
329     {
330     \__tag_role_read_namespace_line:nw {#1} ##1,,, \q_stop
331     }
332     \ior_close:N\g_tmpa_ior
333     }
334     {
335     \msg_info:nnn {tag}{namespace-missing}{#2}
336     }
337 }
338

```

(End of definition for __tag_role_read_namespace:nn.)

__tag_role_read_namespace:n

This command reads the default namespace file.

```

339 \cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
340 {
341     \__tag_role_read_namespace:nn {#1}{#1}
342 }

```

(End of definition for __tag_role_read_namespace:n.)

1.5 Reading the default data

The order is important as we want pdf2 and latex as default: if two namespace define the same tag, the last one defines which one is used if the namespace is not explicitly given.

```

343 \__tag_role_read_namespace:n {pdf}
344 \__tag_role_read_namespace:n {pdf2}
345 \__tag_role_read_namespace:n {mathml}

```

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

```

346 \bool_set_false:N\l__tag_role_update_bool
347 \__tag_role_read_namespace:n {latex-book}
348 \bool_set_true:N\l__tag_role_update_bool
349 \__tag_role_read_namespace:n {latex}
350 \__tag_role_read_namespace:nn {latex} {latex-lab}
351 \__tag_role_read_namespace:n {pdf}
352 \__tag_role_read_namespace:n {pdf2}

```

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

```

353 \pdf_version_compare:NnTF < {2.0}
354 {
355   \hook_gput_code:nnn {begindocument}{tagpdf}
356   {
357     \bool_lazy_and:nnT
358     {
359       \cs_if_exist_p:N \chapter
360     }
361     {
362       \cs_if_exist_p:N \c@chapter
363     }
364     {
365       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
366       {
367         \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
368       }
369     }
370   }
371 }
372 {
373   \hook_gput_code:nnn {begindocument}{tagpdf}
374   {
375     \bool_lazy_and:nnT
376     {
377       \cs_if_exist_p:N \chapter
378     }
379     {
380       \cs_if_exist_p:N \c@chapter
381     }
382     {
383       \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
384       {
385         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ latex-book }
386         \prop_gput:Nne
387         \g__tag_role_rolemap_prop {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
388       }
389     }

```

```

390     }
391 }

```

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.

```

392 \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` and `\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!

```

393 \cs_new_protected:Npn \_tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3
394 {
395   \intarray_gset:Nnn \g_tag_role_parent_child_intarray
396     { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
397 }

```

(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

```

398 \int_zero:N \l__tag_tmpa_int

```

Open the file depending on the PDF version

```

399 \pdf_version_compare:NnTF < {2.0}
400 {
401   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
402 }
403 {
404   \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
405 }

```

Now the main loop over the file

```

406 \ior_map_inline:Nn \g_tmpa_ior
407 {

```

ignore lines containing only comments

```

408   \tl_if_empty:nF{#1}
409   {

```

count the lines ...

```

410     \int_incr:N\l__tag_tmpa_int

```

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

```

411     \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
412     \int_compare:nNnTF {\l__tag_tmpa_int}=1

```


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

```

413     {
414         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
415         {
416             \prop_gput:Nne\g__tag_role_index_prop
417             {##2}
418             {\int_compare:nNnT{##1}<{10}{0}##1}
419         }
420     }

```

now the data lines.

```

421     {
422         \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }

```

get the name of the child tag from the first column

```

423         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl

```

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

```

424         \prop_get:NVN \g__tag_role_index_prop \l__tag_tmpa_tl \l__tag_tmpb_tl

```

remove column 2+3

```

425         \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
426         \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.

```

427         \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
428         {
429             \exp_args:Nne
430             \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
431         }
432     }
433 }
434 }

```

close the read handle.

```

435 \ior_close:N\g_tmpa_ior

```

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

```

436 \prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
437 \prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
438 \prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
439 \pdf_version_compare:NnTF < {2.0}
440 {
441     \int_step_inline:nn{6}
442     {
443         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
444     }
445 }
446 {
447     \int_step_inline:nn{10}
448     {
449         \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
450     }

```

all mathml tags are currently handled identically

```

451     \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
452     \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
453     \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
454     {
455         \prop_gput:NnV\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
456     }
457     \prop_gput:NnV\g__tag_role_index_prop{math}\l__tag_tmpb_tl
458 }

```

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.

```

459 \tl_new:N \l__tag_parent_child_check_tl
460 \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
461 % #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
462 % #4 tl for state
463 {
464     \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
465     \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
466     \bool_lazy_and:nnTF
467     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
468     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
469     {

```

Get the rule from the intarray

```

470         \tl_set:Ne#4
471         {
472             \intarray_item:Nn
473             \g__tag_role_parent_child_intarray
474             {\l__tag_tmpa_tl\l__tag_tmpb_tl}
475         }

```

If the state is something is wrong ...

```

476         \int_compare:nNnT
477         {#4} = {\prop_item:Nn\c__tag_role_rules_prop{}}
478         {
479             %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.

```

480     }

```

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

```

481     \group_begin:
482     \int_compare:nNnT {\l__tag_tmpa_int*\l__tag_loglevel_int} > { 0 }
483     {
484         \prop_get:NVNF\c__tag_role_rules_num_prop #4 \l__tag_tmpa_tl
485         {
486             \tl_set:Nn \l__tag_tmpa_tl {unknown}

```

```

487     }
488     \tl_set:Nn \l__tag_tmpb_tl {#1}
489     \msg_note:nneee
490     { tag }
491     { role-parent-child }
492     { #1 }
493     { #2 }
494     {
495       #4~(=\l__tag_tmpa_tl')
496       \iow_newline:
497       #3
498     }
499   }
500   \group_end:
501 }
502 {
503   \tl_set:Nn#4 {0}
504   \msg_warning:nneee
505   { tag }
506   {role-parent-child}
507   { #1 }
508   { #2 }
509   { unknown! }
510 }
511 }
512 \cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnnN {VVVN,VVN}

```

(End of definition for __tag_role_get_parent_child_rule:nnnN.)

`__tag_check_parent_child:nnnnN`

This command 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.

```

513 \pdf_version_compare:NnTF < {2.0}
514 {
515   \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5
516     {%#1 parent tag,#2 NS, #3 child tag, #4 NS, #5 tl var
517     {

```

for debugging messages we store the arguments.

```

518     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
519     \prop_put:Nnn \l__tag_role_debug_prop {child} {#3}

```

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

```

520     \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
521     {
522       \tl_set:Nn \l__tag_tmpa_tl {#1}
523     }
524     {
525       \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
526       {
527         \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
528       }
529     }

```



```

577         #3
578     }
579     {
580         \tl_set:Nn #3 {0}
581         \msg_warning:nnee
582         { tag }
583         {role-parent-child}
584         { #1 }
585         { #2 }
586         { unknown! }
587     }
588 }

```

and now the real command.

```

589 \cs_new_protected:Npn \__tag_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS, tl va
590 {
591     \prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
592     \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 rolemap-
ping from the namespace

```

593     \tl_if_empty:nTF {#2}
594     {
595         \tl_set:Nn \l__tag_tmpa_tl {#1}
596     }
597     {
598         \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
599         {
600             \prop_get:cnNTF
601             { g__tag_role_NS_#2_prop }
602             {#1}
603             \l__tag_tmpa_tl
604             {
605                 \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
606                 \tl_if_empty:NT\l__tag_tmpa_tl
607                 {
608                     \tl_set:Nn \l__tag_tmpa_tl {#1}
609                 }
610             }
611             {
612                 \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
613             }
614         }
615         {
616             \msg_warning:nnn { tag } {role-unknown-NS} { #2}
617             \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
618         }
619     }

```

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

```

620     \tl_if_empty:nTF {#4}
621     {
622         \tl_set:Nn \l__tag_tmpb_tl {#3}
623     }

```

```

624     {
625       \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
626       {
627         \prop_get:cnNTF
628         { g__tag_role_NS_#4_prop }
629         {#3}
630         \l__tag_tmpb_tl
631         {
632           \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
633           \tl_if_empty:NT\l__tag_tmpb_tl
634           {
635             \tl_set:Nn \l__tag_tmpb_tl {#3}
636           }
637         }
638         {
639           \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
640         }
641       }
642       {
643         \msg_warning:nnn { tag } {role-unknown-NS} { #4}
644         \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
645       }
646     }

```

and now get the relation

```

647     \bool_lazy_and:nnTF
648     { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
649     { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
650     {
651       \__tag_role_get_parent_child_rule:VVnN
652       \l__tag_tmpa_tl \l__tag_tmpb_tl
653       {Rolemapped~from~'#1/#2'--->~'#3\str_if_empty:nF{#4}{/#4}' }
654       #5
655     }
656     {
657       \tl_set:Nn #5 {0}
658       \msg_warning:nneee
659       { tag }
660       {role-parent-child}
661       { #1 }
662       { #3 }
663       { unknown! }
664     }
665   }
666 }
667 \cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
668 \cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
669 \end{package}

```

(End of definition for __tag_check_parent_child:nnnnN.)

\tag_check_child:nnTF

```

670 <base>\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true}
671 *package)
672 \prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}

```

```

673 {
674   \seq_get:NN\g__tag_struct_stack_seq\l__tag_tpa_tl
675   \__tag_struct_get_parentrole:eNN
676     {\l__tag_tpa_tl}
677     \l__tag_get_parent_tpa_tl
678     \l__tag_get_parent_tpb_tl
679   \__tag_check_parent_child:VVnnN
680     \l__tag_get_parent_tpa_tl
681     \l__tag_get_parent_tpb_tl
682     {#1}{#2}
683     \l__tag_parent_child_check_tl
684   \int_compare:nNnTF { \l__tag_parent_child_check_tl } < {0}
685     {\prg_return_false:}
686     {\prg_return_true:}
687 }

```

(End of definition for `\tag_check_child:nTF`. This function is documented on page 156.)

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
688 \tl_new:N \l__tag_role_remap_tag_tl
689 \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?

```

690 \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.

```

691 \cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap:

```

(End of definition for `__tag_role_remap_id:.`)

1.8 Key-val user interface

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

```

tag_(rolemap-key)
tag-namespace_(rolemap-key) 692 \keys_define:nn { __tag / tag-role }
role_(rolemap-key)          693 {
role-namespace_(rolemap-key) 694   ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
role/new-tag_(setup-key)    695   ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
add-new-tag_(deprecated)   696   ,role .tl_set:N = \l__tag_role_role_tmpa_tl
                             697   ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
                             698 }
                             699
700 \keys_define:nn { __tag / setup }
701 {
702   role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
703   ,role/new-tag .code:n =
704   {
705     \keys_set_known:nnnN
706     {__tag/tag-role}
707     {
708       tag-namespace=user,
709       role-namespace=, %so that we can test for it.
710       #1
711       }{__tag/tag-role}\l_tmpa_tl
712     \tl_if_empty:NF \l_tmpa_tl
713     {
714       \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
715       \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
716       \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
717     }
718     \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
719     {
720       \prop_get:NVNTF
721       \g__tag_role_tags_NS_prop
722       \l__tag_role_role_tmpa_tl
723       \l__tag_role_role_namespace_tmpa_tl
724       {
725         \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
726         {
727           \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
728         }
729       }
730       {
731         \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
732       }
733     }
734     \pdf_version_compare:NnTF < {2.0}
735     {
736       %TODO add check for emptyness?
737       \__tag_role_add_tag:VV
738       \l__tag_role_tag_tmpa_tl
739       \l__tag_role_role_tmpa_tl
740     }
741     {
742       \__tag_role_add_tag:VVVV
743       \l__tag_role_tag_tmpa_tl
744       \l__tag_role_tag_namespace_tmpa_tl

```



```

745         \l__tag_role_role_tmpa_tl
746         \l__tag_role_role_namespace_tmpa_tl
747     }
748 }
749 ,role/map-tags .choice:
750 ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-
tags} }
751 ,role/map-tags/pdf .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-
tags} }

```

deprecated names

```

752 , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
753 , add-new-tag .meta:n = {role/new-tag={#1}}
754 }
755 \</package>

```

(End of definition for tag (rolemap-key) and others. These functions are documented on page 156.)

Part X

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

`activate/space_ (setup-key)`
`interwordspace_ (deprecated)`

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`. The old name of the key `interwordspace` is still supported but deprecated.

`show-spaces_ (deprecated)`

This key is deprecated. Use `debug/show=spaces` instead. This key works only with luatex and shows with small red bars where spaces have been inserted. This is only for debugging and is not completely reliable (and change affect other literals and tagging), so it should be used with care.

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2024-06-20} {0.99c}
4 {part of tagpdf - code related to real space chars}
5 </header>
```

1 Code for interword spaces

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

`activate/spaces_ (setup-key)`
`interwordspace_ (deprecated)`
`show-spaces_ (deprecated)`

```
6 <*package>
7 \keys_define:nn { __tag / setup }
8 {
9   activate/spaces .choice:,
10  activate/spaces/true .code:n =
11    { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
12  activate/spaces/false .code:n=
13    { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
14  activate/spaces .default:n = true,
15  debug/show/spaces .code:n = {\bool_set_true:N \l__tag_showspaces_bool},
16  debug/show/spacesOff .code:n = {\bool_set_false:N \l__tag_showspaces_bool},
```

deprecated versions:

```
17   interwordspace .choices:nn = {true,on}{\keys_set:nn{__tag/setup}{activate/spaces={true}}},
18   interwordspace .choices:nn = {false,off}{\keys_set:nn{__tag/setup}{activate/spaces={false}}},
19   interwordspace .default:n = {true},
20   show-spaces .choice:,
```

```

21   show-spaces/true .meta:n = {debug/show=spaces},
22   show-spaces/false .meta:n = {debug/show=spacesOff},
23   show-spaces .default:n = true
24 }
25 \sys_if_engine_pdftex:T
26 {
27   \sys_if_output_pdf:TF
28   {
29     \pdfglyphtounicode{space}{0020}
30     \keys_define:nn { __tag / setup }
31     {
32       activate/spaces/true .code:n = { \AddToHook{shipout/firstpage}[tagpdf/space]{\p
33       activate/spaces/false .code:n = { \RemoveFromHook{shipout/firstpage}[tagpdf/spac
34       activate/spaces .default:n = true,
35     }
36   }
37   {
38     \keys_define:nn { __tag / setup }
39     {
40       activate/spaces .choices:nn = { true, false }
41       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
42       activate/spaces .default:n = true,
43     }
44   }
45 }
46
47
48 \sys_if_engine luatex:T
49 {
50   \keys_define:nn { __tag / setup }
51   {
52     activate/spaces .choice:,
53     activate/spaces/true .code:n =
54     {
55       \bool_gset_true:N \g__tag_active_space_bool
56       \lua_now:e{!tx.__tag.func.markspaceon()}
57     },
58     activate/spaces/false .code:n =
59     {
60       \bool_gset_false:N \g__tag_active_space_bool
61       \lua_now:e{!tx.__tag.func.markspaceoff()}
62     },
63     activate/spaces .default:n = true,
64     debug/show/spaces .code:n =
65     { \lua_now:e{!tx.__tag.trace.showspace=true} },
66     debug/show/spacesOff .code:n =
67     { \lua_now:e{!tx.__tag.trace.showspace=nil} },
68   }
69 }

```

(End of definition for activate/spaces (setup-key), interwordspace (deprecated), and show-spaces (deprecated). These functions are documented on page 6.)

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

```

70 \sys_if_engine_luatex:T
71 {
72   \cs_new_protected:Nn \__tag_fakespace:
73   {
74     \group_begin:
75     \lua_now:e{!tx.__tag.func.fakespace()}
76     \skip_horizontal:n{\c_zero_skip}
77     \group_end:
78   }
79 }

```

We need also a command to interrupt the insertion of real space chars in places where we want to insert manually special spaces. In pdftex this can be done with `\pdfinterwordspaceoff` and `\pdfinterwordspaceon`. These commands insert what-sits and this mean they act globally. In luatex a attribute is used to this effect, for consistency this is also set globally.

The off command sets the attributes in luatex.

```

\tag_spacechar_on: 80 \cs_new_protected:Npn \tag_spacechar_off: {}
\tag_spacechar_off: 81 \cs_new_protected:Npn \tag_spacechar_on: {}
82
83 \sys_if_engine_luatex:T
84 {
85   \cs_set_protected:Npn \tag_spacechar_off:
86   {
87     \lua_now:e
88     {
89       tex.setattribute
90       (
91         "global",
92         luatexbase.attributes.g__tag_interwordspaceOff_attr,
93         1
94       )
95     }
96   }
97   \cs_set_protected:Npn \tag_spacechar_on:
98   {
99     \lua_now:e
100    {
101      tex.setattribute
102      (
103        "global",
104        luatexbase.attributes.g__tag_interwordspaceOff_attr,
105        -2147483647
106      )
107    }
108  }
109 }
110 \sys_if_engine_pdftex:T
111 {
112   \sys_if_output_pdf:T
113   {
114     \cs_set_protected:Npn \tag_spacechar_off:
115     {

```

```
116         \pdfinterwordsoff
117     }
118     \cs_set_protected:Npn \tag_spacechar_on:
119     {
120         \pdfinterwordson
121     }
122 }
123 }
```

```
124 \endpackage
```

(End of definition for `_tag_fakespace:`, `\tag_spacechar_on:`, and `\tag_spacechar_off:`. These functions are documented on page ??.)

Index

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

Symbols	
<code>\#</code>	1030, 1034
<code>\\</code>	10, 23, 27, 28, 44, 45, 46, 51, 53, 55, 62, 65, 67, 73, 75, 86, 256, 257, 258, 405, 468, 476
<code>_</code>	422, 433
A	
<code>activate_</code> (<code>setup-key</code>)	36, <u>255</u>
<code>activate-all_</code> (<code>deprecated</code>)	6
<code>activate-mc_</code> (<code>deprecated</code>)	6
<code>activate-struct_</code> (<code>deprecated</code>)	6
<code>activate-tree_</code> (<code>deprecated</code>)	6
<code>activate/all_</code> (<code>setup-key</code>)	6, <u>259</u>
<code>activate/mc_</code> (<code>setup-key</code>)	6, <u>259</u>
<code>activate/socket_</code> (<code>setup-key</code>)	<u>255</u>
<code>activate/space_</code> (<code>setup-key</code>)	178
<code>activate/spaces_</code> (<code>setup-key</code>)	6, 6
<code>activate/struct_</code> (<code>setup-key</code>)	6, <u>259</u>
<code>activate/struct-dest_</code> (<code>setup-key</code>)	6, <u>259</u>
<code>activate/tagunmarked_</code> (<code>setup-key</code>)	6, <u>290</u>
<code>activate/tree_</code> (<code>setup-key</code>)	6, <u>259</u>
<code>actualtext_</code> (<code>mc-key</code>)	71, <u>255</u> , <u>453</u>
<code>actualtext_</code> (<code>struct-key</code>)	102, <u>487</u>
<code>add-new-tag_</code> (<code>deprecated</code>)	692
<code>add-new-tag_</code> (<code>setup-key</code>)	156
<code>\AddToHook</code>	13, 16, 32, 58, 112, 273, 357, 379, 510, 512, 513, 517, 521, 528, 557, 606
<code>AF_</code> (<code>struct-key</code>)	102, <u>598</u>
<code>AFinline_</code> (<code>struct-key</code>)	102, <u>598</u>
<code>AFinline-o_</code> (<code>struct-key</code>)	102, <u>598</u>
<code>AFref_</code> (<code>struct-key</code>)	102, <u>598</u>
<code>alt_</code> (<code>mc-key</code>)	71, <u>255</u> , <u>453</u>
<code>alt_</code> (<code>struct-key</code>)	101, <u>487</u>
<code>artifact_</code> (<code>mc-key</code>)	71, <u>255</u> , <u>453</u>
artifact-bool internal commands:	
<code>__artifact-bool</code>	121
artifact-type internal commands:	
<code>__artifact-type</code>	121
<code>attr-unknown</code>	20, 79
<code>attribute_</code> (<code>struct-key</code>)	103, <u>1183</u>
<code>attribute-class_</code> (<code>struct-key</code>)	103, <u>1149</u>
B	
benchmark commands:	
<code>\benchmark_tic</code> :	485, 487
<code>\benchmark_toc</code> :	488
bool commands:	
<code>\bool_gset_eq:NN</code> ...	626, 641, 653, 671
<code>\bool_gset_false:N</code>	50, 51, 60, 238, 441, 627, 654
<code>\bool_gset_true:N</code>	47, 49, 55, 135, 177, 369
<code>\bool_if:NTF</code> .	9, 13, 18, 27, 36, 40, 67, 69, 74, 79, 114, 192, 200, 210, 223, 234, 241, 258, 266, 277, 303, 311, 320, 343, 373, 381, 390, 406, 419, 430, 442, 444, 461, 469, 494, 501, 561, 621, 636, 648, 666, 836, 897
<code>\bool_if:nTF</code>	6, 355
<code>\bool_if_exist_p:N</code>	44
<code>\bool_lazy_all:nTF</code>	111
<code>\bool_lazy_and:nnTF</code>	43, 145, 155, 275, 311, 357, 367, 375, 466, 493, 540, 570, 647
<code>\bool_lazy_and_p:nn</code>	8
<code>\bool_new:N</code>	16, 20, 21, 41, 42, 63, 73, 130, 131, 132, 133, 134, 136, 138, 140, 141, 257, 293, 294, 617
<code>\bool_set_false:N</code>	16, 178, 210, 211, 212, 233, 234, 235, 239, 346, 391, 594, 620, 647
<code>\bool_set_true:N</code>	15, 137, 139, 220, 221, 222, 243, 244, 245, 258, 348, 390, 593
<code>\box</code>	376
box commands:	
<code>\box_dp:N</code>	176, 180
<code>\box_ht:N</code>	166
<code>\box_new:N</code>	125, 126
<code>\box_set_dp:Nn</code>	174, 176
<code>\box_set_eq:NN</code>	189
<code>\box_set_ht:Nn</code>	173, 175
<code>\box_use_drop:N</code>	178, 182
<code>\boxmaxdepth</code>	82, 177
C	
c@g internal commands:	
<code>\c@g__tag_MCID_abs_int</code>	11, 15, 24, 33, 46, 53, 55, 64, 70, 105, 134, 180, 181, 239, 242, 269, 274, 303, 344, 351, 416
<code>\c@g__tag_parenttree_obj_int</code>	154, 438
<code>\c@g__tag_struct_abs_int</code>	6, 18, 39, 53, 90,

106, 113, 114, 148, 172, 175, 229,
355, 463, 498, 520, 532, 546, 562,
570, 583, 593, 612, 615, 620, 654,
656, 661, 673, 675, 680, 734, 745,
746, 747, 748, 749, 750, 752, 754,
760, 765, 772, 775, 792, 799, 817,
826, 834, 863, 871, 876, 891, 892,
894, 905, 1003, 1066, 1176, 1179, 1227

cctab commands:
`\c_document_cctab` 73
`\chapter` 167, 359, 377

clist commands:
`\clist_const:Nn` 127, 128
`\clist_if_empty:NTF` 1188
`\clist_map_inline:nn` 151, 577
`\clist_new:N` 123
`\clist_set:Nn` 1153, 1187

color commands:
`\color_select:n` 422, 433

cs commands:
`\cs_generate_variant:Nn` 40, 72, 73,
78, 82, 94, 103, 105, 126, 143, 144,
145, 146, 147, 148, 149, 150, 151,
155, 163, 164, 165, 166, 170, 175,
186, 192, 197, 198, 199, 200, 201,
202, 224, 228, 240, 255, 264, 264,
274, 321, 332, 512, 599, 627, 647,
667, 668, 1082, 1091, 1106, 1116, 1137
`\cs_gset_eq:NN`
280, 868, 869, 1000, 1001, 1063, 1064
`\cs_if_exist:NTF` 485, 563, 608
`\cs_if_exist_p:N` . 9, 359, 362, 377, 380
`\cs_if_exist_use:NTF` 362, 1085
`\cs_if_free:NTF` 47, 70
`\cs_new:Nn` 80,
81, 107, 129, 134, 349, 368, 369, 370
`\cs_new:Npn` 9, 15, 26, 68, 83,
87, 98, 103, 138, 158, 166, 193, 231,
254, 353, 441, 449, 455, 461, 1078, 1117
`\cs_new_eq:NN` 152, 153, 154, 155
`\cs_new_protected:Nn` 72, 127, 169, 352
`\cs_new_protected:Npn` 13, 19, 20, 22,
30, 30, 35, 41, 41, 57, 59, 59, 60, 65,
74, 74, 78, 78, 78, 79, 80, 80, 80, 81,
85, 85, 87, 89, 95, 95, 104, 107, 117,
125, 140, 146, 149, 150, 156, 160,
162, 163, 163, 170, 171, 171, 176,
178, 185, 187, 191, 201, 205, 220,
225, 226, 239, 239, 241, 247, 252,
253, 254, 254, 255, 256, 256, 257,
261, 265, 266, 267, 275, 277, 279,
282, 294, 301, 311, 311, 314, 315,
316, 319, 320, 322, 323, 327, 333,
337, 339, 341, 341, 346, 351, 355,
371, 375, 382, 389, 393, 396, 403,
416, 416, 424, 427, 431, 439, 446,
460, 481, 482, 483, 515, 559, 565,
589, 600, 601, 602, 603, 604, 618,
628, 632, 645, 661, 690, 734, 735,
736, 957, 1014, 1083, 1094, 1107, 1131
`\cs_set:Nn` 686, 687
`\cs_set:Npn` 44, 49
`\cs_set_eq:NN` 14,
20, 76, 77, 78, 98, 187, 188, 189,
190, 191, 192, 193, 194, 195, 227,
228, 230, 231, 232, 233, 348, 349,
350, 351, 487, 488, 679, 680, 681,
682, 688, 689, 691, 693, 694, 695,
696, 865, 866, 997, 998, 1060, 1061
`\cs_set_protected:Nn`
. 171, 233, 282, 428, 434, 915, 916
`\cs_set_protected:Npn`
. 9, 15, 16, 22, 29, 36, 37,
55, 62, 62, 62, 63, 66, 67, 70, 72, 77,
82, 85, 88, 97, 101, 114, 118, 141,
206, 206, 213, 215, 226, 229, 237,
238, 244, 245, 329, 333, 337, 341,
355, 359, 361, 738, 739, 951, 959, 1016
`\cs_to_str:N`
. 12, 18, 25, 32, 39, 58, 59, 65, 66

D

`debug/log␣(setup-key)` 6, 277
`debug/show␣(setup-key)` 276
`debug/structures␣(show-key)` 37, 224
`debug/uncompress␣(setup-key)` 277
`\DeclareOption` 49, 50, 51

dim commands:
`\c_max_dim` 165, 190
`\c_zero_dim` 173, 174, 175

`\documentclass` 22
`\DocumentMetadata` 21

E

`E␣(struct-key)` 102, 487
`\endinput` 28
`\ERRORusetaggingsocket` 89
`exclude-header-footer␣(deprecated)` 674
`\ExecuteOptions` 52

exp commands:
`\exp_args:Ne` 118, 443
`\exp_args:NNe` 82, 85, 99, 191, 211
`\exp_args:Nne` 79, 340, 344, 429, 429, 492
`\exp_args:NNne` 99
`\exp_args:NNno` 714
`\exp_args:NV` 196, 202, 347, 376, 387, 392
`\exp_last_unbraced:NV`
. 184, 185, 240, 241, 454, 458, 940
`\exp_not:n` 308

F	
file commands:	
\file_if_exist:nTF	324
\file_input:n	307
flag commands:	
\flag_clear:n	236
\flag_height:n	185, 248
\flag_new:n	183
\flag_raise:n	249
\fontencoding	6
\fontfamily	6
\fontseries	6
\fontshape	6
\fontsize	6
\footins	566
G	
group commands:	
\group_begin:	66, 74, 175, 367, 481, 605, 695, 703, 744
\group_end:	73, 77, 230, 419, 500, 623, 699, 707, 911
H	
\hangindent	374
\hbox	365
hbox commands:	
\hbox_set:Nn	167, 168
hook commands:	
\hook_gput_code:nmn	7, 11, 33, 57, 65, 79, 155, 236, 258, 259, 355, 373, 379, 383, 702, 715, 725, 738
\hook_new:n	340
\hook_use:n	345
I	
\ignorespaces	36
int commands:	
\int_abs:n	143
\int_case:nnTF	82, 289
\int_compare:nNnTF	22, 57, 68, 97, 112, 118, 132, 132, 169, 172, 180, 210, 211, 229, 256, 259, 284, 290, 377, 384, 391, 398, 401, 412, 418, 418, 426, 433, 441, 448, 476, 482, 534, 543, 684, 783, 849, 995, 1058
\int_compare:nTF	179, 324, 1169, 1171, 1173, 1197, 1223
\int_compare_p:nNn	498
\int_decr:N	217, 240
\int_eval:n	76, 80, 85, 90, 134, 190, 291, 308, 388, 469, 477, 495, 500, 503, 620, 661, 680, 746, 747, 748, 749, 750, 863, 891, 892, 894, 905, 1179
\int_gincr:N	180, 239, 269, 313, 317, 321, 325, 331, 335, 339, 343, 344, 351, 438, 606, 734, 745
\int_gset:Nn	7, 81, 157, 287
\int_gzero:N	295
\int_if_zero:nTF	217, 218, 240, 241, 465, 473
\int_incr:N	92, 209, 232, 410
\int_new:N	77, 124, 129, 205, 263, 296, 297, 298, 299, 598
\int_rand:n	61, 62, 64, 66, 68, 70, 71
\int_set:Nn	278, 281, 284, 285, 286
\int_step_inline:nn	441, 447
\int_step_inline:nmn	25, 90, 229
\int_step_inline:nmnn	148, 173, 176, 193, 309, 315
\int_to_arabic:n	143, 145
\int_to_Hex:n	61, 62, 64, 66, 68, 70, 71
\int_use:N	11, 15, 18, 24, 33, 39, 46, 53, 53, 55, 64, 70, 73, 83, 89, 104, 105, 106, 130, 132, 172, 179, 181, 208, 225, 231, 234, 242, 248, 251, 266, 274, 303, 355, 416, 422, 433, 463, 520, 532, 539, 540, 546, 548, 549, 562, 570, 583, 593, 609, 612, 615, 654, 656, 673, 675, 754, 760, 765, 772, 775, 799, 817, 826, 871, 876, 1003, 1066, 1117, 1176, 1227
\int_zero:N	89, 104, 398
intarray commands:	
\intarray_gset:Nnn	287, 395
\intarray_item:Nn	289, 292, 472
\intarray_new:Nn	279, 392
interwordspace _␣ (deprecated)	178, 6
ior commands:	
\ior_close:N	332, 435
\ior_map_inline:Nn	328, 406
\ior_open:Nn	326, 401, 404
\g_tmpa_ior	326, 328, 332, 401, 404, 406, 435
iow commands:	
\iow_newline:	201, 296, 496
\iow_now:Nn	99
\iow_term:n	181, 184, 190, 194, 194, 282, 347, 351, 355, 359, 363, 367, 371
K	
kernel internal commands:	
_kernel_pdf_object_id_indexed:nm	87
_kernel_pdfdict_name:n	44
keys commands:	
\keys_define:nm	7, 30, 33, 38, 50, 99, 111, 121,

173, 216, 225, 255, 260, 261, 387, 396, 403, 409, 454, 487, 597, 648, 674, 692, 700, 710, 1138, 1149, 1183	87, 87, 94, 99, 105, 109, 109, 118, 122, 130, 131, 136, 142, 156, 183, 230, 247, 261, 269, 285, 306, 320, 330
<code>\keys_set:nn</code> 10, 17, 18, 18, 96, 187, 266, 341, 345, 372, 493, 770	
<code>\keys_set_known:nnnN</code> 705	
L	
<code>label_(mc-key)</code> 71, 255, 453	
<code>label_(struct-key)</code> 101, 487	
<code>lang_(struct-key)</code> 102, 487	
legacy commands:	
<code>\legacy_if:nTF</code> 97, 463, 466, 467	
<code>\llap</code> 422	
<code>log_(deprecated)</code> 277	
ltx. internal commands:	
<code>ltx.__tag.func.alloctag</code> 272	
<code>ltx.__tag.func.fakespace</code> 451	
<code>ltx.__tag.func.fill_parent_tree_- line</code> 821	
<code>ltx.__tag.func.get_num_from</code> . . . 281	
<code>ltx.__tag.func.get_tag_from</code> . . . 300	
<code>ltx.__tag.func.mark_page_- elements</code> 652	
<code>ltx.__tag.func.mark_shipout</code> . . . 804	
<code>ltx.__tag.func.markspaceoff</code> . . . 517	
<code>ltx.__tag.func.markspaceon</code> . . . 517	
<code>ltx.__tag.func.mc_insert_kids</code> . . . 589	
<code>ltx.__tag.func.mc_num_of_kids</code> . . 330	
<code>ltx.__tag.func.output_num_from</code> . . 281	
<code>ltx.__tag.func.output_parenttree</code> 821	
<code>ltx.__tag.func.output_tag_from</code> . . 300	
<code>ltx.__tag.func.space_chars_- shipout</code> 549	
<code>ltx.__tag.func.store_mc_data</code> . . 315	
<code>ltx.__tag.func.store_mc_in_page</code> 633	
<code>ltx.__tag.func.store_mc_kid</code> . . . 324	
<code>ltx.__tag.func.store_mc_label</code> . . 320	
<code>ltx.__tag.func.store_struct_- mcabs</code> 621	
<code>ltx.__tag.func.update_mc_- attributes</code> 641	
<code>ltx.__tag.tables.role_tag_- attribute</code> 270	
<code>ltx.__tag.trace.log</code> 184	
<code>ltx.__tag.trace.show_all_mc_data</code> 241	
<code>ltx.__tag.trace.show_mc_data</code> . . 226	
<code>ltx.__tag.trace.show_prop</code> 201	
<code>ltx.__tag.trace.show_seq</code> 192	
<code>ltx.__tag.trace.show_struct_data</code> 247	
lua commands:	
<code>\lua_now:n</code> 8, 12, 15, 18, 25, 26, 32, 35, 39, 42, 46, 50, 51, 56, 58, 59, 59, 61, 65, 65, 66, 67, 71, 75, 84, 85,	
M	
<code>mathml</code> 102	
<code>\maxdimen</code> 188	
<code>mc-current</code> 19, 16	
<code>mc-current_(show-key)</code> 37, 111	
<code>mc-data_(show-key)</code> 37, 99	
<code>mc-label-unknown</code> 19, 9	
<code>mc-marks_(show-key)</code> 37, 173	
<code>mc-nested</code> 19, 6	
<code>mc-not-open</code> 19, 13	
<code>mc-popped</code> 19, 14	
<code>mc-pushed</code> 19, 14	
<code>mc-tag-missing</code> 19, 8	
<code>mc-used-twice</code> 19, 12	
<code>\MessageBreak</code> 15, 19, 20, 21	
msg commands:	
<code>\msg_error:nn</code> 168, 189, 413, 789	
<code>\msg_error:nnn</code> 205, 216, 224, 235, 270, 400, 1163, 1203	
<code>\msg_error:nnnnn</code> 536, 545	
<code>\msg_info:nnn</code> 134, 174, 182, 258, 262, 327, 335	
<code>\msg_info:nnnn</code> 212, 231	
<code>\msg_line_context:</code> 86, 372, 373, 405, 409, 413, 469, 477	
<code>\g_msg_module_name_prop</code> 30, 34	
<code>\g_msg_module_type_prop</code> 33	
<code>\msg_new:nnn</code> 7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 49, 60, 69, 80, 81, 82, 83, 84, 85, 87, 89, 90, 91, 92, 93, 94, 96, 254, 372, 373, 403, 407, 411, 463, 471	
<code>\msg_new:nnnn</code> 99	
<code>\msg_note:nn</code> 28, 169	
<code>\msg_note:nnn</code> 208, 225, 393, 400, 435, 443	
<code>\msg_note:nnnn</code> 231, 248, 379, 386, 420, 428	
<code>\msg_note:nnnnn</code> 489	
<code>\msg_redirect_name:nnn</code> 532	
<code>\msg_show_item_unbraced:n</code> 246	
<code>\msg_show_item_unbraced:nn</code> 237	
<code>\msg_term:nnnnnn</code> 231, 240	
<code>\msg_warning:nn</code> 24, 215, 261	
<code>\msg_warning:nnn</code> 11, 13, 41, 44, 53, 175, 198, 243, 244, 250, 251, 274, 297, 323, 616, 643, 1009, 1028, 1072	
<code>\msg_warning:nnnn</code> 381, 456, 502	
<code>\msg_warning:nnnnn</code> 217, 407, 504, 551, 581, 658, 856	

N	
namespace _□ (rolemap-key)	156
new-tag	20, 89
newattribute _□ (deprecated)	103, 1131
\newcommand	590, 591
\newcounter	6, 8, 154
\NewDocumentCommand	6, 23, 29, 34, 40, 46, 51, 56, 94, 286, 595
\newmarks	13
no-struct-dest _□ (deprecated)	6
\noindent	374
\nointerlineskip	181
P	
\PackageError	13
\PackageWarning	28, 554
page/exclude-header-footer _□ (setup- key)	39, 674
page/tabsorder _□ (setup-key)	6, 293
para-flattened _□ (deprecated)	387
para-hook-count-wrong	20, 99
para/flattened _□ (tool-key)	387
para/maintag _□ (setup-key)	387
para/maintag _□ (tool-key)	387
para/tag _□ (setup-key)	387
para/tag _□ (tool-key)	387
para/tagging _□ (setup-key)	38, 387
para/tagging _□ (tool-key)	387
\PARALABEL	487
paratag _□ (deprecated)	387
paratagging _□ (deprecated)	38, 387
paratagging-show _□ (deprecated)	38, 387
parent _□ (struct-key)	101, 487
pdf commands:	
\pdf_activate_indexed_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	98, 108, 113, 156, 165, 192, 271, 1134, 1157, 1193
\pdf_object_if_exist:n	142
\pdf_object_if_exist:nTF	652, 714
\pdf_object_new:n	102, 34, 36, 72, 76, 153, 255, 302, 313
\pdf_object_new_indexed:nn	30, 70, 74, 751
\pdf_object_ref:n	102, 56, 85, 130, 130, 134, 143, 194, 310, 327, 654, 716
\pdf_object_ref_indexed:nn	56, 73, 83, 95, 175, 204, 231, 377, 435, 879, 976, 1039, 1080
\pdf_object_ref_last:	102, 103, 117, 123, 254, 1212
\pdf_object_unnamed_write:nn	99, 110, 119, 246, 1207
\pdf_object_write:nnn	73, 80, 250, 274, 303, 322, 329, 334
\pdf_object_write_indexed:nnnn	78, 82, 138, 390
\pdf_pageobject_ref:n	199, 426
\pdf_string_from_unicode:nnN	42
\pdf_uncompress:	287, 289
\pdf_version_compare:NnTF	20, 81, 125, 148, 156, 229, 259, 316, 353, 399, 439, 513, 734
pdfannot commands:	
\pdfannot_dict_put:nnn	144, 709, 732, 750, 755
\pdfannot_link_ref_last:	719, 742
pdfdict commands:	
\pdfdict_gput:nnn	38, 45, 53, 189, 269, 326
\pdfdict_if_empty:nTF	320
\pdfdict_new:n	18, 35, 37
\pdfdict_put:nnn	696, 697, 704, 705
\pdfdict_use:n	276, 324, 331
\pdffakespace	38, 284
pdffile commands:	
\pdffile_embed_stream:nnN	599, 607
\pdffile_embed_stream:nnn	145
\pdfglyphtounicode	29
\pdfinterwordsoff	180, 116
\pdfinterwordsoff	180, 32, 120
pdfmanagement commands:	
\pdfmanagement_add:nnn	51, 69, 70, 295, 297, 299, 385
\pdfmanagement_if_active_p:	9, 10
\pdfmanagement_remove:nn	301
pdfmanagement internal commands:	
\l_pdfmanagement_delayed_ shipout_bool	44, 45
prg commands:	
\prg_do_nothing:	78, 85, 280, 348, 349, 350, 351, 693, 694, 695, 696
\prg_generate_conditional_ variant:Nnn	142
\prg_new_conditional:Nnn	66, 221
\prg_new_conditional:Npnn	105, 128, 143, 153, 347, 353, 364
\prg_new_eq_conditional:NNn	80, 228
\prg_new_protected_conditional:Npnn	670

<code>\prg_replicate:nn</code>	142	<code>\property_ref:nn</code>	101, 155
<code>\prg_return_false</code> : 76, 106, 123, 134, 137, 150, 160, 225, 350, 362, 368, 685		<code>\property_ref:mnn</code>	154
<code>\prg_return_true</code> : ... 77, 120, 133, 147, 157, 224, 351, 361, 367, 670, 686		<code>\providecommand</code> .	62, 63, 64, 291, 559, 560
<code>\prg_set_conditional:Npnn</code>	109	<code>\ProvidesExplFile</code>	3
<code>\prg_set_protected_conditional:Npnn</code>	672	<code>\ProvidesExplPackage</code>	3, 3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 26, 37, 1122
<code>\ProcessOptions</code>	53	Q	
prop commands:		<code>\quad</code>	203, 204
<code>\prop_clear:N</code>	175	quark commands:	
<code>\prop_count:N</code>	196	<code>\q_no_value</code>	527, 537, 612, 617, 639, 644
<code>\prop_get:NnN</code> 137, 145, 179, 200, 213, 216, 270, 301, 403, 424, 436, 438, 451, 452, 464, 465, 567, 568, 851, 1097		<code>\quark_if_no_value:NTF</code>	138, 146, 180, 201, 217, 271, 302, 855, 1104
<code>\prop_get:NnNTF</code>	43, 92, 160, 166, 179, 193, 198, 208, 227, 238, 279, 375, 484, 520, 525, 530, 535, 600, 627, 720, 811, 941, 1023	<code>\quark_if_no_value_p:N</code>	467, 468, 541, 542, 571, 572, 648, 649
<code>\prop_gput:Nnn</code>	26, 30, 31, 33, 34, 56, 88, 89, 90, 91, 92, 94, 97, 97, 98, 99, 99, 100, 110, 111, 112, 113, 119, 120, 121, 122, 144, 146, 148, 151, 163, 190, 203, 207, 219, 222, 262, 284, 284, 287, 288, 316, 328, 377, 378, 385, 386, 416, 437, 443, 449, 455, 457, 893, 904, 978, 1041, 1133, 1165, 1212	<code>\q_stop</code>	261, 294, 330
<code>\prop_gremove:Nn</code>	136, 136	R	
<code>\prop_gset_eq:NN</code>	135, 890	<code>raw_(mc-key)</code>	71, 255, 453
<code>\prop_if_exist:NTF</code>	176, 202, 236, 322, 373, 598, 625, 963, 1020	<code>ref_(struct-key)</code>	102, 487
<code>\prop_if_exist_p:N</code>	495	<code>\RemoveFromHook</code>	33, 515, 516
<code>\prop_if_in:NnTF</code>	70, 130, 165, 173, 272, 725, 1161, 1201, 1205	<code>\renewcommand</code>	593, 594
<code>\prop_item:Nn</code>	41, 74, 148, 185, 193, 221, 292, 396, 446, 470, 477, 478, 901, 1210, 1217	<code>\RenewDocumentCommand</code>	8
<code>\prop_map_function:NN</code>	235	<code>\RequirePackage</code> 20, 54, 313, 316, 322, 325, 555
<code>\prop_map_inline:Nn</code>	260, 265, 286, 318, 358, 365, 383, 453	<code>\rlap</code>	433
<code>\prop_map_tokens:Nn</code>	336	<code>role_(rolemap-key)</code>	156, 692
<code>\prop_new:N</code> 7, 8, 9, 10, 11, 11, 19, 24, 25, 32, 33, 120, 133, 187, 747, 1126, 1129	<code>role-missing</code>	20, 81
<code>\prop_new_linked:N</code> 17, 65, 67, 188, 1127		<code>role-namespace_(rolemap-key)</code> ..	156, 692
<code>\prop_put:Nnn</code> 147, 182, 518, 519, 591, 592	<code>role-parent-child</code>	20, 85
<code>\prop_show:N</code> 64, 91, 195, 887, 908, 1179, 1206	<code>role-remapping</code>	20, 87
property commands:		<code>role-tag</code>	20, 89
<code>\property_gset:mnnn</code>	153	<code>role-unknown</code>	20, 81
<code>\property_new:mnnn</code>	152	<code>role-unknown-NS</code>	20, 81
<code>\property_record:mn</code>	159	<code>role-unknown-tag</code>	20, 81
		<code>role/new-attribute_(setup-key)</code>	103, 1131
		<code>role/new-tag_(setup-key)</code>	692
		<code>root-AF_(setup-key)</code>	103, 710
		S	
		<code>\selectfont</code>	6
		seq commands:	
		<code>\seq_clear:N</code>	279, 314
		<code>\seq_const_from_clist:Nn</code>	21, 34
		<code>\seq_count:N</code>	22, 25, 57, 291, 385, 1169, 1171, 1173, 1197, 1223
		<code>\seq_get:NN</code>	674
		<code>\seq_get:NNTF</code> .	409, 450, 785, 930, 937
		<code>\seq_gpop:NN</code>	923
		<code>\seq_gpop:NNTF</code>	105, 924
		<code>\seq_gpop_left:NN</code>	267
		<code>\seq_gpush:Nn</code> .	13, 15, 88, 95, 792, 832
		<code>\seq_gput_left:Nn</code>	271
		<code>\seq_gput_right:Nn</code> 38, 144, 150, 191, 213, 233, 256, 334

`\seq_gset_eq:NN` 155, 217, 286
`\seq_if_empty:NTF` 196, 379
`\seq_item:Nn` 58, 112, 114, 121, 125,
132, 136, 192, 308, 315, 328, 357,
359, 366, 471, 472, 479, 480, 715, 716
`\seq_log:N` . 171, 195, 219, 263, 421, 436
`\seq_map_function:NN` 244
`\seq_map_indexed_inline:Nn` . 414, 427
`\seq_map_inline:Nn` 280, 282, 1159, 1199
`\seq_new:N` 12, 14, 14, 15, 16, 17, 17,
18, 18, 24, 121, 122, 134, 189, 750, 1130
`\seq_pop_left:NN` 423, 425, 426
`\seq_put_right:Nn` 281
`\seq_remove_all:Nn` 284
`\seq_set_eq:NN` 203, 204
`\seq_set_from_clist:NN` . . . 1154, 1190
`\seq_set_from_clist:Nn`
. 83, 86, 192, 212, 411, 422
`\seq_set_map_e:NNn` 1155, 1191
`\seq_set_split:Nnn`
. 50, 149, 470, 478, 714
`\seq_show:N` . 57, 186, 187, 194, 220,
282, 283, 285, 344, 835, 888, 909, 919
`\seq_use:Nn`
. 45, 106, 107, 201, 203, 204, 343, 1170
`\l_tmpa_seq` 314, 334, 344, 714, 715, 716
`\setbox` 364
shipout commands:
`\g_shipout_readonly_int`
. 104, 179, 234, 388
`show-kids` 20, 59
`show-spaces□(deprecated)` 178, 6
`show-struct` 20, 59
`\ShowTagging` 17, 37, 93
skip commands:
`\skip_horizontal:n` 76
`\c_zero_skip` 76
socket commands:
`\socket_assign_plug:n`
. 486, 508, 509, 525, 750, 751
`\socket_new:n` 437, 438, 467
`\socket_new_plug:n`
. 440, 459, 468, 476, 492
`\socket_use:n` 510, 512, 519, 523
`\socket_use:n` 510
`stash□(mc-key)` 71, 121
`stash□(struct-key)` 101, 487
str commands:
`\str_case:nnTF` 60, 804
`\str_const:Nn` 59
`\str_if_empty:NTF` 653
`\str_if_eq:nnTF` . . . 123, 366, 452, 569
`\str_if_eq_p:n` 312, 357, 359
`\str_new:N` 119
`\str_set_convert:Nnn` . . 150, 278,
299, 467, 480, 514, 526, 540, 556, 587
`\str_use:N` 289, 312
`\c_tilde_str` 52, 54
`\string` 20, 21, 22, 579
`struct-faulty-nesting` 20, 32
`struct-label-unknown` 20, 38
`struct-missing-tag` 20, 35
`struct-no-objnum` 20, 24
`struct-orphan` 20, 25
`struct-show-closing` 20, 40
`struct-stack□(show-key)` 37, 216
`struct-unknown` 20, 22
`struct-used-twice` 20, 36
`\SuspendTagging` 40
sys commands:
`\c_sys_backend_str` 60
`\c_sys_engine_str` 11, 13
`\sys_if_engine luatex:TF` . . . 48,
49, 70, 83, 83, 103, 105, 116, 284, 305
`\sys_if_engine pdftex:TF` . . . 25, 110
`\sys_if_output_pdf:TF` . . . 11, 27, 112
`sys-no-interwordspace` 20, 96

T

`tabsorder□(deprecated)` 6, 293
`tag□(mc-key)` 71, 255, 453
`tag□(rolemap-key)` 156, 692
`tag□(struct-key)` 101, 487
tag commands:
`\tag_check_benchmark_on:` 483
`\tag_check_child:nn` . . . 156, 670, 672
`\tag_check_child:nnTF` 156, 670
`\tag_get:n` 17, 72,
100, 101, 116, 117, 88, 91, 103, 103, 405
`\tag_if_active:` 105, 109
`\tag_if_active:TF` . . . 17, 18, 104, 530
`\tag_if_active_p:` 17, 104, 368
`\tag_if_box_tagged:N` 17, 128
`\tag_if_box_tagged:NTF` 17, 127
`\tag_if_box_tagged_p:N` 17, 127
`\tag_mc_artifact_group_begin:n` . .
. 70, 59, 59, 62
`\tag_mc_artifact_group_end:`
. 70, 59, 60, 70
`\tag_mc_begin:n` . . . 10, 70, 25, 65,
113, 171, 171, 351, 351, 355, 361,
421, 432, 456, 488, 629, 657, 708, 731
`\tag_mc_begin_pop:n` 70,
75, 79, 80, 101, 638, 668, 722, 745
`\tag_mc_end:` 70,
31, 74, 92, 233, 233, 351, 352, 423,
428, 434, 434, 498, 635, 664, 720, 743

`\tag_mc_end_push:`
 . 70, 64, 79, 79, 82, 623, 650, 706, 729
`\tag_mc_if_in:` 80, 228
`\tag_mc_if_in:TF` 70, 42, 66, 221
`\tag_mc_if_in:p:` 70, 66, 221
`\tag_mc_reset_box:N` 71, 78, 78, 245, 245
`\tag_mc_use:n` 70, 35, 35, 36, 37
`\l_tag_para_attr_class_tl` .. 381, 383
`\tag_socket_use:n` .. 39, 40, 62, 66, 67
`\tag_socket_use:nn` . 39, 40, 63, 66, 72
`\tag_spacechar_off:` .. 80, 80, 85, 114
`\tag_spacechar_on:` ... 80, 81, 97, 118
`\tag_start:` 6, 204, 215, 228, 253
`\tag_start:n`
 ... 6, 72, 204, 238, 257, 375, 634, 663
`\tag_stop:` ... 6, 48, 204, 206, 227, 252
`\tag_stop:n`
 ... 6, 67, 204, 229, 256, 373, 630, 658
`\tag_struct_begin:n`
 100, 48, 447, 454, 472,
 482, 656, 707, 730, 734, 734, 738, 739
`\tag_struct_end:`
 100, 26, 53, 500, 504,
 665, 721, 734, 735, 744, 915, 916, 954
`\tag_struct_end:n` 100, 736, 951
`\tag_struct_gput:nnn`
 101, 1083, 1083, 1091
`\tag_struct_insert_annot:nn`
 . 100, 130, 719, 742, 1107, 1107, 1116
`\tag_struct_object_ref:n`
 100, 1077, 1078, 1082
`\tag_struct_parent_int:` 100,
 130, 712, 719, 735, 742, 1107, 1117
`\tag_struct_use:n`
 100, 101, 58, 957, 957, 959
`\tag_struct_use_num:n`
 100, 1014, 1014, 1016
`\tag_tool:n` 36, 13, 13, 14, 16, 20
tag internal commands:
 `__tag_activate_mark_space` 517
 `\g_tag_active_mc_bool`
 40, 114, 130, 145, 262, 269
 `\l_tag_active_mc_bool`
 117, 136, 145, 211, 221, 234, 244
 `\l_tag_active_socket_bool` .. 69,
 74, 79, 136, 212, 222, 235, 245, 263
 `\g_tag_active_space_bool`
 13, 55, 60, 130
 `\g_tag_active_struct_bool`
 113, 130, 155, 264, 271, 277, 406
 `\l_tag_active_struct_bool`
 116, 136, 155, 210, 220, 233, 243
 `\g_tag_active_struct_dest_bool` .
 130, 268, 275, 276
 `\g_tag_active_tree_bool`
 9, 67, 115, 130, 263, 270, 343, 381
 `__tag_add_missing_mcs:Nn`
 83, 163, 163, 215
 `__tag_add_missing_mcs_to_-`
 stream:Nn 65,
 65, 185, 185, 566, 570, 575, 582, 584
 `\g_tag_attr_class_used_prop` ...
 284, 286, 1125, 1165
 `\g_tag_attr_class_used_seq` 282, 1130
 `\g_tag_attr_entries_prop`
 293, 1125, 1133, 1161, 1201, 1206, 1210
 `__tag_attr_new_entry:nn`
 644, 1131, 1131, 1137, 1142, 1146
 `\g_tag_attr_objref_prop`
 1125, 1205, 1212, 1217
 `\l_tag_attr_value_tl` 1125,
 1195, 1214, 1219, 1221, 1225, 1229
 `__tag_backend_create_bdc_node` .. 396
 `__tag_backend_create_bmc_node` .. 367
 `__tag_backend_create_emc_node` .. 338
 `__tag_check_add_tag_role:nn` ...
 129, 201, 201
 `__tag_check_add_tag_role:nnn` ...
 171, 220
 `__tag_check_benchmark_tic:` 348,
 352, 356, 360, 364, 368, 372, 481, 487
 `__tag_check_benchmark_toc:` 350,
 354, 358, 362, 366, 370, 374, 482, 488
 `__tag_check_if_active_mc:` 143
 `__tag_check_if_active_mc:TF` ...
 84, 103,
 142, 173, 187, 235, 357, 363, 430, 436
 `__tag_check_if_active_struct:` . 153
 `__tag_check_if_active_struct:TF`
 39, 142,
 741, 742, 920, 921, 953, 961, 1018, 1110
 `__tag_check_if_mc_in_galley:` .. 347
 `__tag_check_if_mc_in_galley:TF` .
 179, 200
 `__tag_check_if_mc_tmb_missing:` 353
 `__tag_check_if_mc_tmb_missing:TF`
 108, 188, 205, 353
 `__tag_check_if_mc_tmb_missing_-`
 p: 353
 `__tag_check_if_mc_tme_missing:` 364
 `__tag_check_if_mc_tme_missing:TF`
 151, 192, 209, 364
 `__tag_check_if_mc_tme_missing_-`
 p: 364
 `__tag_check_info_closing_-`
 struct:n 178, 178, 186, 926
 `__tag_check_init_mc_used:`
 277, 277, 280, 286

<code>__tag_check_mc_if_nested:</code>	<code>\g__tag_delayed_shipout_bool</code>
. 176, 239, 239, 368 42, 47, 51, 234
<code>__tag_check_mc_if_open:</code>	<code>__tag_exclude_headfoot_begin:</code>
. 237, 239, 247, 440 618, 679, 680
<code>__tag_check_mc_in_galley:TF</code> 347	<code>__tag_exclude_headfoot_end:</code>
<code>__tag_check_mc_in_galley_p:</code> 347 632, 681, 682
<code>__tag_check_mc_pushed_popped:nn</code>	<code>__tag_exclude_struct_headfoot_-</code>
. 89, 96, 109, 112, 117, 254, 254	<code>begin:n</code> 645, 686, 687
<code>__tag_check_mc_tag:N</code>	<code>__tag_exclude_struct_headfoot_-</code>
. 189, 266, 266, 380	<code>end:</code> 661, 688, 689
<code>__tag_check_mc_used:n</code>	<code>__tag_fakespace</code> 451
. 143, 282, 282, 324	<code>__tag_fakespace:</code> 70, 72, 288
<code>\g__tag_check_mc_used_intarray</code>	<code>__tag_finish_structure:</code>
. 277, 287, 289, 292 13, 16, 340, 341
<code>__tag_check_no_open_struct:</code>	<code>__tag_get_data_mc_counter:</code> 9, 9
. 187, 187, 928, 935	<code>__tag_get_data_mc_tag:</code>
<code>__tag_check_para_begin_show:nn</code> 254, 254, 349, 349
. 416, 455, 487	<code>__tag_get_data_struct_counter:</code>
<code>__tag_check_para_end_show:nn</code> 460, 461
. 427, 499	<code>__tag_get_data_struct_id:</code> 449, 449
<code>__tag_check_parent_child:nnN</code>	<code>__tag_get_data_struct_num:</code> 454, 455
. 559, 565, 667	<code>__tag_get_data_struct_tag:</code> 441, 441
<code>__tag_check_parent_child:nnnnN</code> 513	<code>__tag_get_mathsubtype</code> 262
<code>__tag_check_parent_child:nnnnN</code>	<code>__tag_get_mc_abs_cnt:</code>
. 206, 396, 515, 14, 15, 19, 20,
561, 574, 589, 668, 679, 843, 989, 1052	100, 105, 135, 146, 185, 227, 243,
<code>__tag_check_show_MCID_by_page:</code>	251, 263, 270, 271, 289, 310, 324, 334
. 301, 301	<code>__tag_get_mc_cnt_type_tag</code> 256
<code>__tag_check_struct_used:n</code>	<code>__tag_get_num_from</code> 281
. 191, 191, 966	<code>\l__tag_get_parent_tmpa_tl</code>
<code>__tag_check_structure_has_tag:n</code> 117, 204, 207, 220,
. 163, 163, 775	394, 397, 410, 677, 680, 841, 844, 859
<code>__tag_check_structure_tag:N</code>	<code>\l__tag_get_parent_tmpa_tl_uuuu\l_-</code>
. 171, 171, 473, 484	<code>__tag_get_parent_tmpb_tl_uuuu\l_-</code>
<code>__tag_check_timeout_v:n</code> 98, 98,	<code>__tag_tmpa_str</code> 114
106, 107, 110, 145, 153, 160, 198,	<code>\l__tag_get_parent_tmpb_tl</code>
207, 282, 465, 481, 497, 569, 574, 579 118, 205, 208, 220,
<code>__tag_debug_mc_begin_ignore:n</code>	395, 398, 410, 678, 681, 842, 845, 859
. 382, 423	<code>__tag_get_tag_from</code> 300
<code>__tag_debug_mc_begin_insert:n</code>	<code>\l__tag_get_tmpe_tl</code>
. 365, 375 114, 166, 171, 182, 184,
<code>__tag_debug_mc_end_ignore:</code> 396, 448	185, 238, 240, 241, 814, 820, 1100, 1104
<code>__tag_debug_mc_end_insert:</code> 389, 438	<code>__tag_gincr_para_begin_int:</code>
<code>__tag_debug_struct_begin_-</code> 311, 315, 333, 349, 362, 453, 480
<code>ignore:n</code> 424, 913	<code>__tag_gincr_para_end_int:</code>
<code>__tag_debug_struct_begin_-</code> 311, 323, 341, 351, 496
<code>insert:n</code> 416, 910	<code>__tag_gincr_para_main_begin_-</code>
<code>__tag_debug_struct_end_check:n</code>	<code>int:</code> 311, 311, 329, 348, 446, 471
. 446, 953	<code>__tag_gincr_para_main_end_int:</code>
<code>__tag_debug_struct_end_ignore:</code> 311, 319, 337, 350, 503
. 439, 948	<code>__tag_hook_kernel_after_foot:</code>
<code>__tag_debug_struct_end_insert:</code> 604, 613, 682, 689, 696
. 431, 946	<code>__tag_hook_kernel_after_head:</code>
 602, 611, 681, 688, 695

__tag_hook_kernel_before_foot: .
 603, 612, 680, 687, 694
 __tag_hook_kernel_before_head: .
 601, 610, 679, 686, 693
 \g__tag_in_mc_bool
 16, 18, 177, 223, 238,
 369, 441, 626, 627, 641, 653, 654, 671
 __tag_insert_bdc_node 396
 __tag_insert_bmc_node 367
 __tag_insert_emc_node 338
 __tag_lastpagelabel: ... 94, 95, 113
 __tag_log 184
 \l__tag_loglevel_int 129, 132, 169,
 172, 180, 210, 229, 257, 260, 278,
 281, 284, 284, 285, 286, 377, 384,
 391, 398, 418, 426, 433, 441, 448, 482
 __tag_mark_spaces 456
 __tag_mc_artifact_begin_marks:n
 19, 41, 77, 377
 \l__tag_mc_artifact_bool
 20, 124, 178, 192, 239, 373
 \l__tag_mc_artifact_type_tl
 19, 128, 132, 136,
 140, 144, 148, 152, 156, 347, 375, 377
 __tag_mc_bdc:nn 229, 232, 264, 306, 339
 __tag_mc_bdc_mcid:n .. 119, 234, 311
 __tag_mc_bdc_mcid:nn
 234, 237, 267, 313, 318
 __tag_mc_bdc_shipout:nn ... 233, 245
 __tag_mc_begin_marks:nn
 19, 19, 40, 76, 384
 __tag_mc_bmc:n 229, 230, 335
 __tag_mc_bmc_artifact: 333, 333, 346
 __tag_mc_bmc_artifact:n 333, 337, 347
 \l__tag_mc_botmarks_seq
 83, 17, 86, 107,
 157, 187, 204, 204, 212, 217, 349, 366
 __tag_mc_disable_marks: 74, 74
 __tag_mc_emc: 154, 229, 231, 443
 __tag_mc_end_marks: . 19, 59, 78, 444
 \l__tag_mc_firstmarks_seq
 82, 17, 83, 106, 186, 192,
 195, 196, 203, 203, 204, 349, 357, 359
 \g__tag_mc_footnote_marks_seq ... 14
 __tag_mc_get_marks: 80, 80, 178, 199
 __tag_mc_handle_artifact:N
 115, 333, 341, 375
 __tag_mc_handle_mc_label:n
 26, 26, 197, 388
 __tag_mc_handle_mcid:nn
 234, 316, 321, 381
 __tag_mc_handle_stash:n 49, 138,
 140, 141, 170, 227, 322, 322, 332, 416
 __tag_mc_if_in: 66, 80, 221, 228
 __tag_mc_if_in:TF 66, 86, 221, 241, 249
 __tag_mc_if_in_p: 66, 221
 __tag_mc_insert_extra_tmb:n ...
 104, 104, 167
 __tag_mc_insert_extra_tme:n ...
 104, 149, 168
 __tag_mc_insert_mcid_kids:n ...
 129, 129, 148, 269
 __tag_mc_insert_mcid_single_
 kids:n 129, 134, 270
 \l__tag_mc_key_label_tl
 . 22, 194, 197, 319, 384, 385, 388, 489
 \l__tag_mc_key_properties_tl ...
 22, 179, 268, 283, 284,
 304, 305, 383, 463, 472, 473, 485, 486
 \l__tag_mc_key_stash_bool
 20, 27, 36, 123, 200, 390
 \g__tag_mc_key_tag_tl ... 19, 22,
 182, 242, 254, 260, 349, 371, 442, 459
 \l__tag_mc_key_tag_tl 22, 181, 189,
 191, 241, 259, 370, 380, 382, 384, 458
 __tag_mc_lua_set_mc_type_attr:n
 81, 81, 105, 191
 __tag_mc_lua_unset_mc_type_
 attr: 81, 107, 240
 \g__tag_mc_main_marks_seq 14
 \g__tag_mc_marks 13,
 21, 30, 43, 50, 61, 67, 84, 87, 193, 213
 \g__tag_mc_multicol_marks_seq ... 14
 \g__tag_mc_parenttree_prop
 17, 18, 99, 164, 185, 328
 \l__tag_mc_ref_abspage_tl
 11, 270, 282, 290, 298
 __tag_mc_set_label_used:n 30, 30, 50
 \g__tag_mc_stack_seq
 18, 88, 95, 105, 263
 __tag_mc_store:nnn . 89, 89, 103, 130
 \l__tag_mc_tmpa_tl .. 12, 284, 287, 291
 g__tag_MCID_abs_int 7
 \g__tag_MCID_byabspage_prop
 262, 280, 289, 297
 \g__tag_MCID_tmp_bypage_int
 263, 266, 287, 295, 308
 \g__tag_mode_lua_bool
 ... 41, 49, 50, 114, 241, 277, 303,
 311, 320, 369, 561, 621, 636, 648, 666
 __tag_new_output_prop_handler:n
 68, 78, 102, 748
 __tag_pairs_prop 201
 \l__tag_para_attr_class_tl
 292, 383, 485
 \g__tag_para_begin_int
 292, 317, 335, 422, 543, 548

`\l__tag_para_bool` [292](#), [389](#), [398](#), [405](#),
[411](#), [442](#), [461](#), [494](#), [593](#), [594](#), [620](#), [647](#)
`\g__tag_para_end_int`
..... [292](#), [325](#), [343](#), [433](#), [543](#), [549](#)
`\l__tag_para_flattened_bool`
..... [292](#), [394](#), [401](#), [414](#), [444](#), [469](#), [501](#)
`\l__tag_para_main_attr_class_tl` .
..... [292](#), [475](#)
`\g__tag_para_main_begin_int`
..... [292](#), [313](#), [331](#), [534](#), [539](#)
`\g__tag_para_main_end_int`
..... [292](#), [321](#), [339](#), [534](#), [540](#)
`__tag_para_main_store_struct:` ..
..... [353](#), [353](#), [451](#), [477](#)
`\g__tag_para_main_struct_tl` [292](#), [355](#)
`\l__tag_para_main_tag_tl`
..... [292](#), [393](#), [400](#), [413](#), [449](#), [474](#)
`\l__tag_para_show_bool`
..... [292](#), [390](#), [391](#), [406](#), [419](#), [430](#)
`\l__tag_para_tag_default_tl` ... [292](#)
`\l__tag_para_tag_tl`
..... [292](#), [361](#), [392](#), [399](#), [407](#), [412](#), [454](#), [484](#)
`\l__tag_parent_child_check_tl` ...
..... [210](#), [211](#), [400](#), [401](#), [459](#),
[683](#), [684](#), [848](#), [849](#), [994](#), [995](#), [1057](#), [1058](#)
`__tag_parenttree_add_objr:mn` ...
..... [162](#), [162](#), [430](#)
`\l__tag_parenttree_content_tl` ...
..... [169](#), [188](#), [200](#), [220](#), [228](#), [249](#), [252](#)
`\g__tag_parenttree_objr_tl`
..... [161](#), [164](#), [249](#)
`__tag_pdf_name_e:n` [98](#), [98](#)
`__tag_pdf_object_ref` [426](#)
`__tag_prop_gput:Nnn`
..... [9](#), [29](#), [90](#), [120](#), [127](#),
[131](#), [187](#), [190](#), [197](#), [288](#), [296](#), [307](#), [972](#)
`__tag_prop_item:Nn` .. [9](#), [49](#), [187](#), [193](#)
`__tag_prop_new:N` [9](#),
[9](#), [11](#), [101](#), [187](#), [187](#), [199](#), [262](#), [746](#)
`__tag_prop_new_linked:N`
..... [15](#), [17](#), [187](#), [188](#)
`__tag_prop_show:N` [9](#), [62](#), [187](#), [195](#), [202](#)
`__tag_property_gset:nnnn`
..... [152](#), [153](#), [265](#)
`\c__tag_property_mc_clist`
..... [127](#), [244](#), [305](#)
`__tag_property_new:nnnn`
..... [152](#), [152](#), [170](#), [173](#), [177](#), [180](#), [184](#)
`__tag_property_record:nn`
..... [28](#), [156](#), [165](#), [240](#), [301](#), [417](#), [779](#)
`__tag_property_ref:nn` . [155](#), [164](#), [580](#)
`__tag_property_ref:nnn`
..... [41](#), [152](#), [154](#), [163](#),
[168](#), [180](#), [184](#), [194](#), [199](#), [200](#), [272](#),
[326](#), [337](#), [426](#), [964](#), [970](#), [973](#), [979](#), [986](#)
`__tag_property_ref_lastpage:nn` .
. [82](#), [159](#), [166](#), [166](#), [173](#), [176](#), [305](#), [319](#)
`\c__tag_property_struct_clist` ...
..... [127](#), [781](#)
`g__tag_role/RoleMap_dict` [18](#)
`\g__tag_role_add_mathml_bool` ...
..... [73](#), [258](#), [702](#), [752](#)
`__tag_role_add_tag:nn`
..... [127](#), [127](#), [155](#), [282](#), [367](#), [737](#)
`__tag_role_add_tag:nnnn`
..... [169](#), [169](#), [228](#), [314](#), [742](#)
`__tag_role_alloctag:nnn` [81](#),
[85](#), [95](#), [107](#), [117](#), [126](#), [142](#), [186](#), [279](#), [310](#)
`\l__tag_role_debug_prop`
..... [157](#), [11](#), [518](#), [519](#), [591](#), [592](#)
`__tag_role_get:nnNN`
..... [156](#), [158](#), [166](#), [229](#), [231](#), [255](#), [481](#), [793](#)
`__tag_role_get_parent_child_-
rule:nnnN` [171](#), [459](#), [460](#), [512](#), [544](#), [651](#)
`\g__tag_role_index_prop` . [157](#), [10](#),
[416](#), [424](#), [436](#), [437](#), [438](#), [443](#), [449](#),
[451](#), [452](#), [455](#), [457](#), [464](#), [465](#), [520](#), [530](#)
`\g__tag_role_NS<ns>_class_prop` [157](#)
`\g__tag_role_NS<ns>_prop` [157](#)
`\g__tag_role_NS_mathml_prop` [260](#), [453](#)
`__tag_role_NS_new:nnn`
. [159](#), [20](#), [22](#), [30](#), [74](#), [75](#), [76](#), [77](#), [78](#), [80](#)
`\g__tag_role_NS_prop`
... [157](#), [9](#), [26](#), [56](#), [166](#), [318](#), [336](#), [725](#)
`\g__tag_role_parent_child_-
intarray` [392](#), [395](#), [473](#)
`__tag_role_read_namespace:n` [339](#),
[339](#), [343](#), [344](#), [345](#), [347](#), [349](#), [351](#), [352](#)
`__tag_role_read_namespace:nn` ...
..... [320](#), [320](#), [341](#), [350](#)
`__tag_role_read_namespace_-
line:nw` [257](#), [261](#), [294](#), [330](#)
`__tag_role_remap:`
..... [690](#), [690](#), [691](#), [867](#), [999](#), [1062](#)
`__tag_role_remap_id:` [691](#), [691](#)
`\l__tag_role_remap_NS_tl`
. [688](#), [866](#), [869](#), [998](#), [1001](#), [1061](#), [1064](#)
`\l__tag_role_remap_tag_tl`
. [688](#), [865](#), [868](#), [997](#), [1000](#), [1060](#), [1063](#)
`\l__tag_role_role_namespace_-
tmpa_tl` [12](#),
[697](#), [718](#), [723](#), [725](#), [727](#), [731](#), [746](#)
`\l__tag_role_role_tmpa_tl`
..... [12](#), [696](#), [716](#), [722](#), [739](#), [745](#)
`\g__tag_role_rolemap_prop`
..... [157](#), [18](#), [145](#), [148](#), [151](#), [160](#),
[216](#), [219](#), [222](#), [262](#), [265](#), [387](#), [525](#), [535](#)

\c__tag_role_rules_num_prop 393, 484
\c__tag_role_rules_prop 393, 396, 477
\l__tag_role_tag_namespace_tmpa_-
tl 12, 567, 571, 575, 695, 744
\l__tag_role_tag_namespace_tmpb_-
tl 14, 568, 569, 572, 576
\l__tag_role_tag_namespace_tmpb_-
tluuuuuu% 12
\l__tag_role_tag_tmpa_tl
..... 12, 694, 715, 738, 743
\g__tag_role_tags_class_prop ...
... 157, 8, 90, 99, 112, 121, 137, 270
\g__tag_role_tags_NS_prop .. 157,
7, 88, 97, 110, 119, 130, 173, 208,
272, 385, 470, 478, 567, 568, 721, 941
\l__tag_role_tmpa_seq 12
\l__tag_role_update_bool
..... 210, 257, 258, 266, 346, 348
\c__tag_role_userNS_id_str
..... 158, 59, 80
\g__tag_root_default_tl 255
\g__tag_saved_in_mc_bool
..... 617, 626, 641, 653, 671
__tag_seq_gput_right:Nn 9,
36, 187, 191, 198, 208, 218, 228, 251
__tag_seq_item:Nn ... 9, 44, 187, 192
__tag_seq_new:N
..... 9, 9, 22, 103, 187, 189, 200, 749
__tag_seq_show:N . 9, 55, 187, 194, 201
__tag_show_spacemark 437
\l__tag_showspaces_bool 15, 16
__tag_space_chars_shipout 549
__tag_start_para_ints:
..... 223, 246, 327, 327
__tag_stop_para_ints:
..... 213, 236, 327, 346
__tag_store_parent_child_-
rule:nnn 393, 393, 430
g__tag_struct_1_prop 100
__tag_struct_add_AF:nn
..... 611, 628, 647, 654, 673, 716
__tag_struct_add_inline_AF:nn ..
..... 600, 627, 687, 691, 698, 706
\g__tag_struct_AFobj_int 598, 606, 609
\g__tag_struct_cont_mc_prop
..... 11, 91, 92, 94, 97, 221
\g__tag_struct_dest_num_prop ... 64
\l__tag_struct_elem_stash_bool ..
..... 63, 490, 837, 897
__tag_struct_exchange_kid_-
command:N 265, 265, 274, 305
__tag_struct_fill_kid_key:n ...
..... 135, 275, 275, 388
__tag_struct_format_parentrole:nn
..... 368, 369
__tag_struct_format_Ref:nn 370, 370
__tag_struct_format_rolemap:nn .
..... 368, 368
__tag_struct_get_dict_content:nN
..... 137, 355, 355, 389
__tag_struct_get_id:n
. 95, 100, 113, 114, 137, 138, 395, 451
__tag_struct_get_parentrole:nNN
..... 176,
176, 192, 202, 392, 675, 839, 985, 1048
__tag_struct_gput_data_ref:nn ..
..... 582, 1093, 1094, 1106
__tag_struct_insert_annot:nn ...
..... 403, 403, 1112
\l__tag_struct_key_label_tl
..... 62, 489, 777, 780
__tag_struct_kid_mc_gput_-
right:nn ... 193, 205, 206, 224, 325
__tag_struct_kid_OBJR_gput_-
right:nnn .. 241, 241, 244, 264, 418
__tag_struct_kid_struct_gput_-
right:nn
... 225, 225, 226, 240, 884, 968, 1031
g__tag_struct_kids_1_seq 100
\l__tag_struct_lang_tl
..... 599, 732, 757, 762
__tag_struct_mcid_dict:n
..... 94, 97, 193, 211
\c__tag_struct_null_tl 10, 309
\g__tag_struct_objR_seq 8
__tag_struct_output_prop_aux:nn
..... 68, 68, 82
__tag_struct_prop_gput:nnn
86, 87, 88, 94, 105, 110, 115, 120,
127, 153, 162, 168, 311, 324, 338,
519, 531, 545, 561, 569, 592, 614,
655, 674, 717, 753, 759, 764, 798,
816, 825, 875, 1035, 1101, 1175, 1226
\g__tag_struct_ref_by_dest_prop . 67
\g__tag_struct_roletag_NS_tl ... 58
\l__tag_struct_roletag_NS_tl ...
..... 61, 797, 802, 829
\l__tag_struct_roletag_tl
..... 58, 796, 802, 804, 829, 833
__tag_struct_set_tag_info:nnn ..
148, 150, 160, 175, 771, 870, 1002, 1065
\g__tag_struct_stack_current_tl .
..... 16, 25, 34, 65, 71, 97, 146,
152, 160, 166, 203, 214, 224, 280,
326, 330, 393, 404, 413, 446, 451,
457, 834, 882, 886, 887, 908, 926,
932, 969, 976, 982, 1032, 1039, 1045

\l__tag_struct_stack_parent_-	271, 271, 273, 277, 279, 281, 288,
tmpa_tl	16, 411, 420, 435, 300, 301, 302, 303, 304, 306, 307,
500, 769, 783, 787, 812, 840, 852,	308, 309, 310, 311, 389, 394, 406,
862, 879, 883, 885, 888, 900, 901, 909	413, 423, 424, 425, 426, 436, 437,
\g__tag_struct_stack_seq	12, 22, 25, 438, 443, 449, 450, 451, 454, 455,
410, 674, 786, 792, 835, 919, 924, 930	458, 464, 467, 474, 481, 482, 484,
\c__tag_struct_StructElem_-	486, 495, 520, 522, 525, 527, 541,
entries_seq	21, 545, 576, 579, 583, 595, 603, 605,
\c__tag_struct_StructTreeRoot_-	606, 608, 610, 612, 613, 617, 648,
entries_seq	21, 652, 674, 676, 854, 855, 862, 923,
\g__tag_struct_tag_NS_tl	58, 924, 930, 932, 937, 940, 941, 943,
472, 480, 481, 483, 774, 795, 847,	987, 992, 1026, 1050, 1055, 1167, 1178
860, 866, 869, 873, 907, 943, 991,	
998, 1001, 1005, 1054, 1061, 1064, 1068	
\g__tag_struct_tag_stack_seq . . .	
.	14, 45, 219, 220, 421, 436, 450, 832, 923, 937
\g__tag_struct_tag_tl	
.	58, 181, 182, 185, 370, 371, 471, 473, 479, 481, 482, 484,
371, 471, 473, 479, 481, 482, 484,	773, 794, 833, 846, 860, 865, 868,
773, 794, 833, 846, 860, 865, 868,	872, 939, 941, 983, 990, 997, 1000,
872, 939, 941, 983, 990, 997, 1000,	1004, 1046, 1053, 1060, 1063, 1067
1004, 1046, 1053, 1060, 1063, 1067	
__tag_struct_write_obj:n	
.	150, 371, 371
\l__tag_tag_stop_int	204, 208, 209, 217, 218, 225, 231, 232, 240, 241, 248
\g__tag_tagunmarked_bool	141, 290, 292
\l__tag_tmpa_box	
.	114, 167, 173, 174, 178, 189, 190
\l__tag_tmpa_clist	
.	114, 1153, 1154, 1187, 1188, 1190
\l__tag_tmpa_int	89, 92, 97, 100, 104, 113, 114, 398, 410, 412, 482
\l__tag_tmpa_prop	
.	114, 175, 183, 196, 198
\l__tag_tmpa_seq	
.	50, 57, 58, 114, 279, 281, 283, 284, 285, 286, 411, 414, 422, 423,
284, 285, 286, 411, 414, 422, 423,	425, 426, 427, 470, 471, 472, 478,
425, 426, 427, 470, 471, 472, 478,	479, 480, 1155, 1159, 1169, 1170,
479, 480, 1155, 1159, 1169, 1170,	1171, 1173, 1191, 1197, 1199, 1223
1171, 1173, 1191, 1197, 1199, 1223	
\l__tag_tmpa_str	42, 43, 48, 119, 279, 284, 289, 300, 305,
42, 48, 119, 279, 284, 289, 300, 305,	312, 468, 473, 481, 486, 515, 522,
312, 468, 473, 481, 486, 515, 522,	527, 534, 541, 548, 557, 564, 588, 595
527, 534, 541, 548, 557, 564, 588, 595	
\l__tag_tmpa_tl	41, 42, 46, 48, 49, 50, 55, 84, 87, 91, 92,
41, 42, 46, 48, 49, 50, 55, 84, 87, 91, 92,	93, 94, 101, 105, 105, 107, 108, 112,
93, 94, 101, 105, 105, 107, 108, 112,	113, 114, 114, 115, 115, 137, 137,
113, 114, 114, 115, 115, 137, 137,	138, 140, 142, 142, 145, 146, 151,
138, 140, 142, 142, 145, 146, 151,	179, 180, 182, 185, 186, 196, 198,
179, 180, 182, 185, 186, 196, 198,	199, 200, 201, 202, 204, 207, 216,
199, 200, 201, 202, 204, 207, 216,	216, 217, 222, 224, 267, 268, 270,
216, 217, 222, 224, 267, 268, 270,	271, 271, 273, 277, 279, 281, 288,
271, 271, 273, 277, 279, 281, 288,	300, 301, 302, 303, 304, 306, 307,
300, 301, 302, 303, 304, 306, 307,	308, 309, 310, 311, 389, 394, 406,
308, 309, 310, 311, 389, 394, 406,	413, 423, 424, 425, 426, 436, 437,
413, 423, 424, 425, 426, 436, 437,	438, 443, 449, 450, 451, 454, 455,
438, 443, 449, 450, 451, 454, 455,	458, 464, 467, 474, 481, 482, 484,
458, 464, 467, 474, 481, 482, 484,	486, 495, 520, 522, 525, 527, 541,
486, 495, 520, 522, 525, 527, 541,	545, 576, 579, 583, 595, 603, 605,
545, 576, 579, 583, 595, 603, 605,	606, 608, 610, 612, 613, 617, 648,
606, 608, 610, 612, 613, 617, 648,	652, 674, 676, 854, 855, 862, 923,
652, 674, 676, 854, 855, 862, 923,	924, 930, 932, 937, 940, 941, 943,
924, 930, 932, 937, 940, 941, 943,	987, 992, 1026, 1050, 1055, 1167, 1178
987, 992, 1026, 1050, 1055, 1167, 1178	
\l__tag_tmpb_box	
.	114, 168, 175, 176, 180, 182
\l__tag_tmpb_seq	
.	114, 1154, 1155, 1190, 1191
\l__tag_tmpb_tl	169, 88, 103, 114, 117, 119, 375, 424,
.	169, 88, 103, 114, 117, 119, 375, 424,
169, 88, 103, 114, 117, 119, 375, 424,	430, 452, 457, 465, 468, 474, 481,
430, 452, 457, 465, 468, 474, 481,	483, 488, 530, 532, 535, 537, 542,
483, 488, 530, 532, 535, 537, 542,	545, 622, 630, 632, 633, 635, 639,
545, 622, 630, 632, 633, 635, 639,	644, 649, 652, 988, 993, 1051, 1056
644, 649, 652, 988, 993, 1051, 1056	
__tag_tree_fill_parenttree: . . .	
.	170, 171, 246
__tag_tree_final_checks:	20, 20, 346
\g__tag_tree_id_pad_int	77, 81, 143
__tag_tree_lua_fill_parenttree:	
.	226, 226, 243
\g__tag_tree_openaction_struct_-	
tl	31, 37, 56
__tag_tree_parenttree_rerun_-	
msg:	170, 213, 248
__tag_tree_update_openaction: . .	
.	41, 74
__tag_tree_write_classmap:	
.	279, 279, 361
__tag_tree_write_idtree:	85, 353
__tag_tree_write_namespaces: . . .	
.	314, 314, 365
__tag_tree_write_parenttree: . . .	
.	239, 239, 349
__tag_tree_write_rolemap:	
.	256, 256, 357
__tag_tree_write_structelements:	
.	146, 146, 369
__tag_tree_write_structtreeroot:	
.	125, 125, 373
__tag_whatsits:	
.	35, 57, 62, 63, 66, 351, 352
tag-namespace_(rolemap-key)	692
tag/struct/1 internal commands:	
__tag/struct/1	30
tag/tree/namespaces internal commands:	
__tag/tree/namespaces	313

tag/tree/parenttree internal commands:	
__tag/tree/parenttree	153
tag/tree/rolemap internal commands:	
__tag/tree/rolemap	255
tagabspage	6, 170
tagmcabs	6, 170
\tagmcbegin	36, 157, 22, 370, 376
\tagmccend	36, 22, 376
tagmccid	6, 170
\tagmccifin	36
\tagmccifinTF	36, 39
\tagmccuse	36, 22
\tagpdfparaOff	38, 590
\tagpdfparaOn	38, 590
\tagpdfsetup	36, 103, 156, 6
\tagpdfsuppressmarks	38, 595
\tagstart	6, 228, 255
\tagstop	6, 227, 254
tagstruct	6, 170
\tagstructbegin	37, 156, 157, 45, 258, 361, 363
\tagstructend	37, 45, 259, 376
tagstructobj	6, 170
\tagstructuse	37, 45
\tagtool	36, 13
tagunmarked _␣ (deprecated)	6, 290
test/lang _␣ (setup-key)	597
\TeX and \LaTeX 2 _ε commands:	
\M	164
\@auxout	99
\@bsphack	158
\@cclv	570
\@esphack	160
\@gobble	31, 55
\@ifpackageloaded	28, 553
\@kernel@after@foot	613
\@kernel@after@head	611
\@kernel@before@cclv	560, 567
\@kernel@before@foot	612
\@kernel@before@footins	563, 565
\@kernel@before@head	608, 610
\@kernel@tag@hangfrom	359
\@kernel@tagssupport@makecol	559, 572
\@makecol	569, 574
\@maxdepth	177
\@mult@ptagging@hook	577
\@outputbox	575
\@secondoftwo	31, 55
\@tempboxa	364, 374, 376
\c@chapter	362, 380
\c@page	569, 574
\count@	582
\mult@firstbox	580
\mult@rightbox	584
\new@label@record	101
\on@line	466, 481, 497
\page@sofar	579
\process@cols	580
tex commands:	
\tex_botmarks:D	87
\tex_firstmarks:D	84
\tex_kern:D	180
\tex_marks:D	21, 30, 43, 50, 61, 67
\tex_special:D	66
\tex_splitbotmarks:D	213
\tex_splitfirstmarks:D	193
texsource	102
\the	569, 574
\tiny	422, 433
title _␣ (struct-key)	101, 487
title-o _␣ (struct-key)	101, 487
tl commands:	
\c_empty_tl	367, 387
\c_space_tl	53, 54, 55, 55, 74, 103, 166, 190, 191, 193, 196, 198, 200, 209, 252, 290, 348, 364, 370, 394, 569, 574, 637, 863, 900, 982, 1045, 1104, 1170, 1216
\tl_clear:N	87, 88, 105, 179, 188, 189, 281, 357, 569, 576
\tl_const:Nn	10
\tl_count:n	78, 82, 143
\tl_gput_right:Nn	164, 635
\tl_gset:Nn	18, 32, 37, 97, 242, 256, 260, 268, 301, 355, 442, 459, 471, 472, 479, 480, 482, 483, 642, 834, 932, 939, 943
\tl_gset_eq:NN	182, 371
\tl_head:N	605, 632
\tl_if_empty:NTF	42, 43, 108, 194, 268, 299, 345, 385, 606, 633, 712, 718, 757, 776
\tl_if_empty:nTF	51, 64, 72, 84, 143, 198, 203, 212, 222, 264, 268, 276, 297, 297, 299, 408, 469, 477, 478, 538, 554, 593, 603, 620, 671
\tl_if_empty_p:n	312
\tl_if_eq:NNTF	309, 349
\tl_if_eq:NnTF	107
\tl_if_eq:nnTF	214, 267, 280
\tl_if_exist:NTF	130, 308, 381, 630
\tl_if_head_eq_charcode:nNTF	48
\tl_if_in:nnTF	187
\tl_new:N	11, 12, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 25, 31, 32, 58, 59, 60, 61, 62, 114, 115, 116, 117, 118, 161, 169, 255, 300, 302, 304, 306, 309, 310, 459, 640, 688, 689, 732, 1128

