

tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer[†]

Released 2021-07-03

Contents

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

*This file describes v0.91, last revised 2021-07-03.

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

2	Description of log messages	13
2.1	\ShowTagging command	13
2.2	Messages in checks and commands	13
2.3	Warning messages from the lua-code	14
2.4	Info messages from the lua-code	14
3	Messages	15
3.1	Messages related to mc-chunks	15
3.2	Messages related to mc-chunks	16
3.3	Attributes	16
3.4	Roles	17
3.5	Miscellaneous	17
4	Retrieving data	17
5	User conditionals	17
6	Internal checks	18
6.1	checks for active tagging	18
6.2	Checks related to stuctures	18
6.3	Checks related to roles	20
6.4	Check related to mc-chunks	20

II The tagpdf-user module

Code related to L^AT_EX2e user commands and document commands

	Part of the tagpdf package	24
1	Setup commands	24
2	Commands related to mc-chunks	24
3	Commands related to structures	24
4	Debugging	25
5	Extension commands	25
5.1	Fake space	25
5.2	Paratagging	25
5.3	Link tagging	26
6	User commands and extensions of document commands	26
7	Setup and preamble commands	26
8	Commands for the mc-chunks	26
9	Commands for the structure	27
10	Debugging	28

11	Commands to extend document commands	30
11.1	Document structure	30
11.2	Fake space	30
11.3	Paratagging	30
11.4	Links	32
III	The <code>tagpdf-tree</code> module	
	Commands trees and main dictionaries	
	Part of the <code>tagpdf</code> package	34
1	Trees, pdfmanagement and finalization code	34
1.1	Catalog: MarkInfo and StructTreeRoot	34
1.2	Writing structure elements	35
1.3	ParentTree	35
1.4	Rolemap dictionary	38
1.5	Classmap dictionary	38
1.6	Namespaces	39
1.7	Finishing the structure	40
1.8	StructParents entry for Page	40
IV	The <code>tagpdf-mc-shared</code> module	
	Code related to Marked Content (mc-chunks), code shared by	
	all modes	
	Part of the <code>tagpdf</code> package	41
1	Public Commands	41
2	Public keys	42
3	Marked content code – shared	42
3.1	Variables and counters	43
3.2	Functions	44
3.3	Keys	46
V	The <code>tagpdf-mc-generic</code> module	
	Code related to Marked Content (mc-chunks), generic mode	
	Part of the <code>tagpdf</code> package	48
1	Marked content code – generic mode	48
1.1	Variables	48
1.2	Functions	48
1.3	Keys	52
VI	The <code>tagpdf-mc-luacode</code> module	
	Code related to Marked Content (mc-chunks), luamode-specific	
	Part of the <code>tagpdf</code> package	54

1	Marked content code – luamode code	54
1.1	Commands	55
1.2	Key definitions	59
VII	The tagpdf-struct module	
	Commands to create the structure	
	Part of the tagpdf package	62
1	Public Commands	62
2	Public keys	62
2.1	Keys for the structure commands	62
2.2	Setup keys	64
3	Variables	64
3.1	Variables used by the keys	66
4	Commands	66
4.1	Initialization of the StructTreeRoot	67
4.2	Handlings kids	68
5	Keys	72
6	User commands	76
7	Attributes and attribute classes	79
7.1	Variables	79
7.2	Commands and keys	79
VIII	The tagpdf-luatex.def	
	Driver for luatex	
	Part of the tagpdf package	82
1	Loading the lua	82
2	Logging functions	86
3	Helper functions	88
3.1	Retrieve data functions	88
3.2	Functions to insert the pdf literals	90
4	Function for the real space chars	91
5	Function for the tagging	94
6	Parenttree	98

IX	The <code>tagpdf-roles</code> module	
	Tags, roles and namespace code	
	Part of the <code>tagpdf</code> package	100
1	Code related to roles and structure names	100
1.1	Variables	100
1.2	Namespaces	101
1.3	Data	102
1.4	Adding new tags and rolemapping	108
1.4.1	pdf 1.7 and earlier	108
1.4.2	The pdf 2.0 version	109
1.5	Key-val user interface	110
X	The <code>tagpdf-space</code> module	
	Code related to real space chars	
	Part of the <code>tagpdf</code> package	112
1	Code for interword spaces	112
	Index	114

1 Initialization and test if pdfmanagement is active.

```
1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2021-07-03} {0.91}
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\DeclareDocumentMetadata{<options>}\MessageBreak
22       before~\string\documentclass
23     }
24 }
```

We map the internal module name “tag” to “tagpdf” in messages.

```
25 \prop_if_exist:NT \g_msg_module_name_prop
26 {
27   \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
28 }
```

2 Package options

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

```
29 \bool_new:N\g__tag_mode_lua_bool
30 \DeclareOption {luamode} { \sys_if_engine luatex:T { \bool_gset_true:N \g__tag_mode_lua_bool } }
31 \DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
32 \ExecuteOptions{luamode}
33 \ProcessOptions
```

3 Packages

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

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

4 Temporary code

This is code which will be removed when proper support exists in LaTeX

4.1 a LastPage label

See also issue #2 in Accessible-xref

`__tag_lastpagelabel:`

```

35 \cs_new_protected:Npn \__tag_lastpagelabel:
36 {
37   \legacy_if:nT { @files }
38   {
39     \exp_args:NNx \exp_args:NNx\iow_now:Nn \@auxout
40     {
41       \token_to_str:N \newlabeldata
42       {__tag_LastPage}
43       {
44         {abspage} { \int_use:N \g_shipout_readonly_int }
45         {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
46       }
47     }
48   }
49 }
50
51 \AddToHook{enddocument/afterlastpage}
52 { \__tag_lastpagelabel: }

```

(End definition for `__tag_lastpagelabel:`)

`\ref_value:nnn` This allows to locally set a default value if the label or the attribute doesn't exist. See issue #4 in Accessible-xref.

```

\ref_value:nnn{<label>}{<attribute>}{<Fallback default>}

53 \cs_if_exist:NF \ref_value:nnn
54 {
55   \cs_new:Npn \ref_value:nnn #1#2#3
56   {
57     \exp_args:Nee
58     \__ref_value:nnn
59     { \tl_to_str:n {#1} } { \tl_to_str:n {#2} } {#3}
60   }
61   \cs_new:Npn \__ref_value:nnn #1#2#3
62   {
63     \tl_if_exist:cTF { g__ref_label_ #1 _ #2 _tl }
64     { \tl_use:c { g__ref_label_ #1 _ #2 _tl } }
65     {
66       #3
67     }
68   }
69 }

```

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

5 Variables

`\l__tag_tmpa_tl` A few temporary variables
`\l__tag_tmpa_str`
`\l__tag_tmpa_prop`
`\l__tag_tmpa_seq`
`\l__tag_tmpb_seq`
`\l__tag_tmpa_clist`
`\l__tag_tmpa_int`

```

70 \tl_new:N \l__tag_tmpa_tl
71 \str_new:N \l__tag_tmpa_str
72 \prop_new:N \l__tag_tmpa_prop
73 \seq_new:N \l__tag_tmpa_seq
74 \seq_new:N \l__tag_tmpb_seq
75 \clist_new:N \l__tag_tmpa_clist
76 \int_new:N \l__tag_tmpa_int

```

(End 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_refmc_clist
\c__tag_refstruct_clist
77 \clist_const:Nn \c__tag_refmc_clist {tagabspage,tagmcabs,tagmcid}
78 \clist_const:Nn \c__tag_refstruct_clist {tagstruct,tagstructobj}

```

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

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

```

79 \int_new:N \l__tag_loglevel_int

```

(End definition for `\l__tag_loglevel_int`.)

`\g__tag_active_space_bool` These booleans should help to control the global behaviour of tagpdf. Ideally it should more or less do nothing if all are false. The space-boolean controles 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. 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.

```

80 \bool_new:N \g__tag_active_space_bool
81 \bool_new:N \g__tag_active_mc_bool
82 \bool_new:N \g__tag_active_tree_bool
83 \bool_new:N \g__tag_active_struct_bool

```

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

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

```

84 \bool_new:N \l__tag_active_mc_bool
85 \bool_set_true:N \l__tag_active_mc_bool
86 \bool_new:N \l__tag_active_struct_bool
87 \bool_set_true:N \l__tag_active_struct_bool

```

(End definition for `\l__tag_active_mc_bool` and `\l__tag_active_struct_bool`.)

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

```

88 \bool_new:N \g__tag_tagunmarked_bool

```

(End definition for `\g__tag_tagunmarked_bool`.)

6 Variants of l3 commands

```

89 \prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F}
90 \cs_generate_variant:Nn \pdf_object_ref:n {e}
91 \cs_generate_variant:Nn \pdfannot_dict_put:nnn {nnx}
92 \cs_generate_variant:Nn \pdffile_embed_stream:nnn {nxx,oxx}
93 \cs_generate_variant:Nn \prop_gput:Nnn {Nxx}
94 \cs_generate_variant:Nn \prop_put:Nnn {Nxx}
95 \cs_generate_variant:Nn \ref_label:nn { nv }
96 \cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
97 \cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }

```

7 Setup label attributes

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

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

```

98 \ref_attribute_gset:nnnn { tagstruct } {0} { now }
99 { \int_use:N \c@g__tag_struct_abs_int }
100 \ref_attribute_gset:nnnn { tagstructobj } {} { now }
101 {
102   \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
103   {
104     \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
105   }
106 }
107 \ref_attribute_gset:nnnn { tagabspage } {0} { shipout }
108 { \int_use:N \g_shipout_readonly_int }
109 \ref_attribute_gset:nnnn { tagmcabs } {0} { now }
110 { \int_use:N \c@g__tag_MCID_abs_int }
111 \ref_attribute_gset:nnnn {tagmcid } {0} { now }
112 { \int_use:N \g__tag_MCID_tmp_bypage_int }

```

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

8 Label commands

`__tag_ref_label:nn` A version of `\ref_label:nn` to set a label which takes a keyword `mc` or `struct` to call the relevant lists. TODO: check if `\@bsphack` and `\@esphack` make sense here.

```

113 \cs_new_protected:Npn \__tag_ref_label:nn #1 #2 %#1 label, #2 name of list mc or struct
114 {
115   \@bsphack
116   \ref_label:nv {#1}{c__tag_ref#2_clist}
117   \@esphack
118 }
119 \cs_generate_variant:Nn \__tag_ref_label:nn {en}

```

(End definition for __tag_ref_label:nn.)

`__tag_ref_value:nnn` A local version to retrieve the value. It is a direct wrapper, but to keep naming consistent It uses the variant defined temporarily above.

```

120 \cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 %#1 label, #2 attribute, #3 default
121 {
122   \ref_value:nnn {#1}{#2}{#3}
123 }
124 \cs_generate_variant:Nn \__tag_ref_value:nnn {enn}

```

(End definition for __tag_ref_value:nnn.)

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

```

125 \cs_new:Npn \__tag_ref_value_lastpage:nn #1 #2
126 {
127   \ref_value:nnn {\__tag_LastPage}{#1}{#2}
128 }

```

(End definition for __tag_ref_value_lastpage:nn.)

9 Commands to fill seq and prop

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

```

\__tag_prop_new:N
\__tag_seq_new:N 129 \cs_set_eq:NN \__tag_prop_new:N \prop_new:N
\__tag_prop_gput:Nnn 130 \cs_set_eq:NN \__tag_seq_new:N \seq_new:N
\__tag_seq_gput_right:Nn 131 \cs_set_eq:NN \__tag_prop_gput:Nnn \prop_gput:Nnn
\__tag_seq_item:cn 132 \cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
\__tag_prop_item:cn 133 \cs_set_eq:NN \__tag_seq_item:cn \seq_item:cn
\__tag_seq_show:N 134 \cs_set_eq:NN \__tag_prop_item:cn \prop_item:cn
\__tag_prop_show:N 135 \cs_set_eq:NN \__tag_seq_show:N \seq_show:N
136 \cs_set_eq:NN \__tag_prop_show:N \prop_show:N
137
138 \cs_generate_variant:Nn \__tag_prop_gput:Nnn { NxN , Nxx, Nnx , cnn, cxn, cnx, cno}
139 \cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Nx , No, cn, cx }
140 \cs_generate_variant:Nn \__tag_prop_new:N { c }
141 \cs_generate_variant:Nn \__tag_seq_new:N { c }
142 \cs_generate_variant:Nn \__tag_seq_show:N { c }
143 \cs_generate_variant:Nn \__tag_prop_show:N { c }

```

(End definition for __tag_prop_new:N and others.)

10 General tagging commands

`\tag_stop_group_begin:` We need a command to stop tagging in some places. This simply switches the two local booleans.

```

144 \cs_new_protected:Npn \tag_stop_group_begin:
145 {
146   \group_begin:
147   \bool_set_false:N \l__tag_active_struct_bool
148   \bool_set_false:N \l__tag_active_mc_bool

```

```

149     }
150     \cs_set_eq:NN \tag_stop_group_end: \group_end:

```

(End definition for \tag_stop_group_begin: and \tag_stop_group_end:. These functions are documented on page ??.)

11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

activate-space Keys to (globally) activate tagging. **activate-space** activates the additional parsing needed for interword spaces. It is not documented, the parsing is currently implicitly activated by the known key **interwordspace**, as the code will perhaps move to some other place, now that it is better separated.

```

activate-mc
activate-tree
activate-struct
activate-all
151 \keys_define:nn { __tag / setup }
152 {
153     activate-space .bool_gset:N = \g__tag_active_space_bool,
154     activate-mc    .bool_gset:N = \g__tag_active_mc_bool,
155     activate-tree  .bool_gset:N = \g__tag_active_tree_bool,
156     activate-struct .bool_gset:N = \g__tag_active_struct_bool,
157     activate-all   .meta:n =
158         {activate-mc,activate-tree,activate-struct},
159

```

(End definition for activate-space and others. These functions are documented on page ??.)

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

```

160     log .choice:,
161     log / none .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
162     log / v .code:n = {\int_set:Nn \l__tag_loglevel_int { 1 }},
163     log / vv .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
164     log / vvv .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
165     log / all .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},

```

(End definition for log. This function is documented on page ??.)

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

```

166     tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
167     tagunmarked .initial:n = true,

```

(End definition for tagunmarked. This function is documented on page ??.)

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

```

168     tabsorder .choice:,
169     tabsorder / row .code:n =
170         \pdfmanagement_add:nnn { Page } {Tabs}{/R},
171     tabsorder / column .code:n =
172         \pdfmanagement_add:nnn { Page } {Tabs}{/C},
173     tabsorder / structure .code:n =

```

```

174     \pdfmanagement_add:nnn { Page } {Tabs}{/S},
175     tabsorder / none       .code:n =
176     \pdfmanagement_remove:nn {Page} {Tabs},
177     tabsorder       .initial:n = structure,
178     uncompress      .code:n = { \pdf_uncompress: },
179 }

```

(End definition for tabsorder. This function is documented on page ??.)

12 loading of engine/more dependent code

```

180 \sys_if_engine luatex:T
181 {
182     \file_input:n {tagpdf-luatex.def}
183 }
184 \</package>
185 \<*/mcloading>
186 \bool_if:NTF \g__tag_mode_lua_bool
187 {
188     \RequirePackage {tagpdf-mc-code-lua}
189 }
190 {
191     \RequirePackage {tagpdf-mc-code-generic} %
192 }
193 \</mcloading>

```

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. Currently the only sensible values for the argument *<keyword>* are `mc_tag` and `struct_tag`.

2 Description of log messages

2.1 \ShowTagging command

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

2.2 Messages in checks and commands

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

2.3 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

2.4 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. **PARENTTREE** is the code building the parenttree.

message	log-level	remark
INFO SHIPOUT-INSERT-LAST-EMC	3	finish of shipout code
INFO SPACE-FUNCTION-FONT	3	interwordspace code
INFO TAG-ABSPAGE	3	
INFO TAG-ARGS	4	
INFO TAG-ENDHEAD	4	
INFO TAG-ENDHEAD	4	
INFO TAG-HEAD	3	
INFO TAG-INSERT-ARTIFACT	3	
INFO TAG-INSERT-BDC	3	
INFO TAG-INSERT-EMC	3	
INFO TAG-INSERT-TAG	3	
INFO TAG-KERN-SUBTYPE	4	
INFO TAG-MATH-SUBTYPE	4	
INFO TAG-MC-COMPARE	4	
INFO TAG-MC-INTO-PAGE	3	
INFO TAG-NEW-MC-NODE	4	
INFO TAG-NODE	3	
INFO TAG-NO-HEAD	3	
INFO TAG-NOT-TAGGED	2	replaced by artifact
INFO TAG-QUITTING-BOX	4	
INFO TAG-STORE-MC-KID	4	
INFO TAG-TRAVERSING-BOX	3	
INFO TAG-USE-ACTUALTEXT	3	
INFO TAG-USE-ALT	3	
INFO TAG-USE-RAW	3	
INFO TEX-MC-INSERT-KID	3	
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	

message	log-level	remark
INFO PARENTTREE-NUM	3	
INFO PARENTTREE-NUMENTRY	3	
INFO PARENTTREE-STRUCT-OBJREF	4	

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-checks-code} {2021-07-03} {0.91}
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 definition for mc-nested. This function is documented on page ??.)

mc-tag-missing If the tag is missing

```

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

mc-pushed Informational messages about mc-pushing.

```

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

```

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

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 definition for mc-current. This function is documented on page 25.)

3.2 Messages related to mc-chunks

struct-no-objnum Should not happen ...

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

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

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

```
23 \msg_new:nnn { tag }  
24 {struct-faulty-nesting}  
25 { there~is~no~open~structure~on~the~stack }
```

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

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

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

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

struct-used-twice

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

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

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

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

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

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

```
31 \msg_new:nnn { tag } {struct-show-closing}  
32 { closing~structure~#1~tagged~\prop_item:cn{g__tag_struct_#1_prop}{S} }
```

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

3.3 Attributes

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

attr-unknown

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

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

3.4 Roles

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

`role-unknown` `34 \msg_new:nnn { tag } {role-missing} { tag~#1~has~no~role~assigned }`

`role-unknown-tag` `35 \msg_new:nnn { tag } {role-unknown} { role~#1~is~not~known }`

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

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

`role-tag` Info messages.

`new-tag` `37 \msg_new:nnn { tag } {role-tag} { mapping~tag~#1~to~role~#2 }`

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

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

3.5 Miscellaneous

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

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

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

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

`sys-no-interwordspace` Currently only pdf_lat_ex and lua_lat_ex have some support for real spaces.

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

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

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

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` and `struct_tag`.

`43 \cs_new:Npn \tag_get:n #1 { \use:c {__tag_get_data_#1:} }`

(End definition for \tag_get:n. This function is documented on page 13.)

5 User conditionals

`\tag_if_active_p:` This is a test 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`

`44 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }`

`45 {`

`46 \bool_lazy_all:nTF`

`47 {`

`48 {\g__tag_active_struct_bool}`

`49 {\g__tag_active_mc_bool}`

`50 {\g__tag_active_tree_bool}`

`51 {\l__tag_active_struct_bool}`

`52 {\l__tag_active_mc_bool}`

`53 }`

```

54     {
55         \prg_return_true:
56     }
57     {
58         \prg_return_false:
59     }
60 }

```

(End definition for `\tag_if_active:TF`. This function is documented on page 13.)

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

```

\__tag_check_if_active_mc:TF Structures must have a tag, so we check if the S entry is in the property. It is an error if
\__tag_check_if_active_struct:TF this is missing. The argument is a number.
61 \prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
62 {
63     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
64     {
65         \prg_return_true:
66     }
67     {
68         \prg_return_false:
69     }
70 }
71 \prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
72 {
73     \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
74     {
75         \prg_return_true:
76     }
77     {
78         \prg_return_false:
79     }
80 }

```

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

6.2 Checks related to structures

```

\__tag_check_structure_has_tag:n Structures must have a tag, so we check if the S entry is in the property. It is an error if
this is missing. The argument is a number. The tests for existence and type is split in
structures, as the tags are stored differently to the mc case.
81 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
82 {
83     \prop_if_in:cnF { g__tag_struct_#1_prop }
84     {S}
85     {
86         \msg_error:nn { tag } {struct-missing-tag}

```

```

87     }
88 }

```

(End definition for `_tag_check_structure_has_tag:n`.)

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

```

89 \cs_new_protected:Npn \_tag\_check\_structure\_tag:N #1
90 {
91   \prop_if_in:Nof \g__tag_role_tags_prop {#1}
92   {
93     \msg_warning:nnx { tag } {role-unknown-tag} {#1}
94   }
95 }

```

(End definition for `_tag_check_structure_tag:N`.)

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

```

96 \cs_new_protected:Npn \_tag\_check\_info\_closing\_struct:n #1 % #1 struct num
97 {
98   \int_compare:nNtT {\l__tag_loglevel_int} > { 0 }
99   {
100     \msg_info:nnn { tag } {struct-show-closing} {#1}
101   }
102 }

```

103

```

104 \cs_generate_variant:Nn \_tag\_check\_info\_closing\_struct:n {o,x}

```

(End definition for `_tag_check_info_closing_struct:n`.)

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

```

105 \cs_new_protected:Npn \_tag\_check\_no\_open\_struct:
106 {
107   \msg_error:nn { tag } {struct-faulty-nesting}
108 }

```

(End definition for `_tag_check_no_open_struct:`.)

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

```

109 \cs_new_protected:Npn \_tag\_check\_struct\_used:n #1 % #1 label
110 {
111   \prop_get:cnNT
112     {g__tag_struct\_tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
113     {P}
114     \l_tmpa_tl
115     {
116       \msg_warning:nnn { tag } {struct-used-twice} {#1}
117     }
118 }

```

(End definition for `_tag_check_struct_used:n`.)

6.3 Checks related to roles

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

```

119 \cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
120 {
121   \tl_if_empty:nTF {#2}
122   {
123     \msg_warning:nnn { tag } {role-missing} {#1}
124   }
125   {
126     \prop_get:NnNTF \g__tag_role_tags_prop {#2} \l_tmpa_tl
127     {
128       \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
129       {
130         \msg_info:nnnn { tag } {role-tag} {#1} {#2}
131       }
132     }
133     {
134       \msg_warning:nnn { tag } {role-unknown} {#2}
135     }
136   }
137 }

```

(End definition for __tag_check_add_tag_role:nn.)

6.4 Check related to mc-chunks

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

```

138 \cs_new_protected:Npn \__tag_check_mc_if_nested:
139 {
140   \__tag_mc_if_in:T
141   {
142     \msg_warning:nnx { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
143   }
144 }
145
146 \cs_new_protected:Npn \__tag_check_mc_if_open:
147 {
148   \__tag_mc_if_in:F
149   {
150     \msg_warning:nnx { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
151   }
152 }

```

(End definition for __tag_check_mc_if_nested: and __tag_check_mc_if_open:.)

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

```

153 \cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
154 {
155   \int_compare:nNnT
156   { \l__tag_loglevel_int } = { 2 }

```

```

157     { \msg_info:nxx {tag}{mc-#1}{#2} }
158 \int_compare:nNtT
159 { \l__tag_loglevel_int } > { 2 }
160 {
161     \msg_info:nxx {tag}{mc-#1}{#2}
162     \seq_log:N \g__tag_mc_stack_seq
163 }
164 }

```

(End definition for __tag_check_mc_pushed_popped:nn.)

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

```

165 \cs_new_protected:Npn \__tag_check_mc_tag:N #1  {%#1 is var with a tag name in it
166 {
167     \tl_if_empty:NT #1
168     {
169         \msg_error:nxx { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
170     }
171     \prop_if_in:Nof \g__tag_role_tags_NS_prop {#1}
172     {
173         \msg_warning:nxx { tag } {role-unknown-tag} {#1}
174     }
175 }

```

(End definition for __tag_check_mc_tag:N.)

\g__tag_check_mc_used_intarray
__tag_check_init_mc_used:

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

```

176 \cs_new_protected:Npn \__tag_check_init_mc_used:
177 {
178     \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
179     \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
180 }

```

(End definition for \g__tag_check_mc_used_intarray and __tag_check_init_mc_used:.)

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

```

181 \cs_new_protected:Npn \__tag_check_mc_used:n #1 {%#1 mcid abs cnt
182 {
183     \int_compare:nNtT {\l__tag_loglevel_int} > { 2 }
184     {
185         \__tag_check_init_mc_used:
186         \intarray_gset:Nnn \g__tag_check_mc_used_intarray
187             {#1}
188             { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
189         \int_compare:nNtT
190         {
191             \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
192         }

```

```

193         >
194         { 1 }
195         {
196             \msg_warning:nnn { tag } {mc-used-twice} {#1}
197         }
198     }
199 }

```

(End definition for _tag_check_mc_used:n.)

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

```

200 \cs_new_protected:Npn \_tag\_check\_show\_MCID\_by\_page:
201 {
202     \tl_set:Nx \l__tag_tmpa_tl
203     {
204         \_tag_ref_value_lastpage:nn
205         {abspage}
206         {-1}
207     }
208     \int_step_inline:nnnn {1}{1}
209     {
210         \l__tag_tmpa_tl
211     }
212     {
213         \seq_clear:N \l_tmpa_seq
214         \int_step_inline:nnnn
215         {1}
216         {1}
217         {
218             \_tag_ref_value_lastpage:nn
219             {tagmcabs}
220             {-1}
221         }
222         {
223             \int_compare:nT
224             {
225                 \_tag_ref_value:enn
226                 {mcid-###1}
227                 {tagabspage}
228                 {-1}
229                 =
230                 ##1
231             }
232             {
233                 \seq_gput_right:Nx \l_tmpa_seq
234                 {
235                     Page##1-###1-
236                     \_tag_ref_value:enn
237                     {mcid-###1}
238                     {tagmcid}
239                     {-1}
240                 }
241             }
242         }

```

```

243         \seq_show:N \l_tmpa_seq
244     }
245 }

(End definition for \_tag_check_show_MCID_by_page:.)

246 \endpackage

```

Part II

The **tagpdf-user** module

Code related to L^AT_EX2e user commands and document commands

Part of the tagpdf package

1 Setup commands

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

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

2 Commands related to mc-chunks

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

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

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

This is a wrapper around `\tag_mc_if_in:TF` and tests if an mc is open or not. It is mostly of importance for pdf_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.

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

\ShowTagging \ShowTagging {<key-val>}

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

mc-data mc-data = <number>

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.

mc-current mc-current

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.

struct-stack struct-stack = log|show

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

5 Extension commands

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

The commands and keys should be view as experimental!

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

5.1 Fake space

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

5.2 Paratagging

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

```

paratagging      paratagging = true|false
paratagging-show paratagging-show = true|false

```

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

```

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

```

5.3 Link tagging

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

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

```

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

```

6 User commands and extensions of document commands

```

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-user} {2021-07-03} {0.91}
4   {tagpdf - user commands}
5 </header>

```

7 Setup and preamble commands

```

\tagpdfsetup
6 <*package>
7 \NewDocumentCommand \tagpdfsetup { m }
8   {
9     \keys_set:nn { __tag / setup } { #1 }
10  }

```

(End definition for `\tagpdfsetup`. This function is documented on page 24.)

8 Commands for the mc-chunks

```

\tagmcbegin
\tagmcend 11 \NewDocumentCommand \tagmcbegin { m }
\tagmcuse 12 {

```

```

13   \tag_mc_begin:n {#1}%\ignorespaces
14   }
15
16
17 \NewDocumentCommand \tagmcend { }
18 {
19   %\if_mode_horizontal: \unskip \fi: %
20   \tag_mc_end:
21 }
22
23 \NewDocumentCommand \tagmcuse { m }
24 {
25   \tag_mc_use:n {#1}
26 }
27

```

(End definition for `\tagmcbegin`, `\tagmcend`, and `\tagmcuse`. These functions are documented on page 24.)

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

```

28 \NewDocumentCommand \tagmcifinTF { m m }
29 {
30   \tag_mc_if_in:TF { #1 } { #2 }
31 }

```

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

9 Commands for the structure

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

```

32 \NewDocumentCommand \tagstructbegin { m }
33 {
34   \tag_struct_begin:n {#1}
35 }
36
37 \NewDocumentCommand \tagstructend { }
38 {
39   \tag_struct_end:
40 }
41
42 \NewDocumentCommand \tagstructuse { m }
43 {
44   \tag_struct_use:n {#1}
45 }

```

(End definition for `\tagstructbegin`, `\tagstructend`, and `\tagstructuse`. These functions are documented on page 24.)

`\tagpdfifluatexTF` I should deprecate them ...

```

\tagpdfifluatexT
\tagpdfifpdfTeXTF
46 \cs_set_eq:NN\tagpdfifluatexTF \sys_if_engine luatex:TF
47 \cs_set_eq:NN\tagpdfifluatexT \sys_if_engine luatex:T
48 \cs_set_eq:NN\tagpdfifpdfTeXT \sys_if_engine pdfTeX:T

```

(End definition for `\tagpdfifluatexTF`, `\tagpdfifluatexT`, and `\tagpdfifpdfTeXTF`. These functions are documented on page ??.)

10 Debugging

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

```

49 \NewDocumentCommand\ShowTagging { m }
50 {
51   \keys_set:nn { __tag / show }{ #1}
52
53 }
```

(End definition for `\ShowTagging`. This function is documented on page 25.)

mc-data 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.

```

54 \keys_define:nn { __tag / show }
55 {
56   mc-data .code:n =
57   {
58     \sys_if_engine_luatex:T
59     {
60       \lua_now:e{!tx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
61     }
62   }
63   ,mc-data .default:n = 1
64 }
65
```

(End definition for `mc-data`. This function is documented on page 25.)

mc-current This shows some info about the current mc-chunk. It works in generic and lua-mode.

```

66 \keys_define:nn { __tag / show }
67 { mc-current .code:n =
68   {
69     \bool_if:NTF \g__tag_mode_lua_bool
70     {
71       \sys_if_engine_luatex:T
72       {
73         \int_compare:nNnTF
74         { -2147483647 }
75         =
76         {
77           \lua_now:e
78           {
79             tex.print
80             (tex.getattribute
81              (luatexbase.attributes.g__tag_mc_cnt_attr))
82           }
83         }
84       }
85     }
86   }
```

```

85         \lua_now:e
86         {
87             ltx.__tag.trace.log
88             (
89                 "mc-current:~no~MC~open,~current~absent
90                 =\__tag_get_mc_abs_cnt:"
91                 ,0
92             )
93             texio.write_nl("")
94         }
95     }
96     {
97         \lua_now:e
98         {
99             ltx.__tag.trace.log
100             (
101                 "mc-current:~absent=\__tag_get_mc_abs_cnt:=="
102                 ..
103                 tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
104                 ..
105                 "~=>tag="
106                 ..
107                 tostring
108                 (ltx.__tag.func.get_tag_from
109                 (tex.getattribute
110                 (luatexbase.attributes.g__tag_mc_type_attr)))
111                 ..
112                 "="
113                 ..
114                 tex.getattribute
115                 (luatexbase.attributes.g__tag_mc_type_attr)
116                 ,0
117             )
118             texio.write_nl("")
119         }
120     }
121 }
122 }
123 {
124     \msg_note:nn{ tag }{ mc-current }
125 }
126 }
127 }

```

(End definition for mc-current. This function is documented on page 25.)

struct-stack

```

128 \keys_define:nn { __tag / show }
129 {
130     struct-stack .choice:
131     ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
132     ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
133     ,struct-stack .default:n = show
134 }

```

(End definition for `struct-stack`. This function is documented on page 25.)

11 Commands to extend document commands

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

11.1 Document structure

```

\__tag_add_document_structure:n
  activate
135 \cs_new_protected:Npn \__tag_add_document_structure:n #1
136 {
137   \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=#1}}
138   \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
139 }
140 \keys_define:nn { __tag / setup}
141 {
142   activate .code:n =
143   {
144     \keys_set:nn { __tag / setup }
145     { activate-mc,activate-tree,activate-struct }
146     \__tag_add_document_structure:n {#1}
147   },
148   activate .default:n = Document
149 }

```

(End definition for `__tag_add_document_structure:n` and `activate`. This function is documented on page ??.)

11.2 Fake space

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

```

150 \sys_if_engine luatex:T
151 {
152   \NewDocumentCommand\pdffakespace { }
153   {
154     \__tag_fakespace:
155   }
156 }

```

(End definition for `\pdffakespace`. This function is documented on page 25.)

11.3 Paratagging

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

`\l__tag_para_bool` At first some variables.

`\l__tag_para_show_bool` 157 `\bool_new:N \l__tag_para_bool`

`\g__tag_para_int` 158 `\bool_new:N \l__tag_para_show_bool`

159 `\int_new:N \g__tag_para_int`

(End definition for `\l__tag_para_bool`, `\l__tag_para_show_bool`, and `\g__tag_para_int`.)

paratagging These keys enable/disable locally paratagging, and the debug modus. It can affect the

paratagging-show typesetting if `paratagging-show` is used. The small numbers are boxes and they have a (small) height.

160 `\keys_define:nn { __tag / setup }`

161 `{`

162 `paratagging .bool_set:N = \l__tag_para_bool,`

163 `paratagging-show .bool_set:N = \l__tag_para_show_bool,`

164 `}`

165

(End definition for `paratagging` and `paratagging-show`. These functions are documented on page 26.)

This fills the para hooks with the needed code.

166 `\AddToHook{para/begin}`

167 `{`

168 `\int_gincr:N \g__tag_para_int`

169 `\bool_if:NT \l__tag_para_bool`

170 `{`

171 `\tag_struct_begin:n {tag=P}`

172 `\bool_if:NT \l__tag_para_show_bool`

173 `{ \tag_mc_begin:n{artifact}`

174 `\llap{\color_select:n{red}\tiny\int_use:N\g__tag_para_int\ }`

175 `\tag_mc_end:`

176 `}`

177 `\tag_mc_begin:n {tag=P}`

178 `}`

179 `}`

180 `\AddToHook{para/end}`

181 `{`

182 `\bool_if:NT \l__tag_para_bool`

183 `{`

184 `\tag_mc_end:`

185 `\bool_if:NT \l__tag_para_show_bool`

186 `{ \tag_mc_begin:n{artifact}`

187 `\rlap{\color_select:n{red}\tiny\int_use:N\g__tag_para_int}`

188 `\tag_mc_end:`

189 `}`

190 `\tag_struct_end:`

191 `}`

192 `}`

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

\tagpdfparaOff is longer.

193 `\newcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}`

194 `\newcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}`

(End definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 26.)

11.4 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.

```
195 \hook_gput_code:nnn
196   {pdfannot/link/URI/before}
197   {tagpdf}
198   {
199     \tag_mc_end_push:
200     \tag_struct_begin:n { tag=Link }
201     \tag_mc_begin:n { tag=Link }
202     \pdfannot_dict_put:nnx
203       { link/URI }
204       { StructParent }
205       { \tag_struct_parent_int: }
206   }
207
208 \hook_gput_code:nnn
209   {pdfannot/link/URI/after}
210   {tagpdf}
211   {
212     \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
213     \tag_mc_end:
214     \tag_struct_end:
215     \tag_mc_begin_pop:n{ }
216   }
217
218 \hook_gput_code:nnn
219   {pdfannot/link/GoTo/before}
220   {tagpdf}
221   {
222     \tag_mc_end_push:
223     \tag_struct_begin:n{tag=Link}
224     \tag_mc_begin:n{tag=Link}
225     \pdfannot_dict_put:nnx
226       { link/GoTo }
227       { StructParent }
228       { \tag_struct_parent_int: }
229   }
230
231 \hook_gput_code:nnn
232   {pdfannot/link/GoTo/after}
233   {tagpdf}
234   {
235     \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
236     \tag_mc_end:
237     \tag_struct_end:
238     \tag_mc_begin_pop:n{ }
239   }
240 }
241
242 % "alternative descriptions " for PAX3. How to get better text here??
243 \pdfannot_dict_put:nnn
```



```
244 { link/URI }
245 { Contents }
246 { (url) }
247
248 \pdfannot_dict_put:nnn
249 { link/GoTo }
250 { Contents }
251 { (ref) }
252
</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} {2021-07-03} {0.91}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
5 </header>
```

1 Trees, pdfmanagement and finalization code

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

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

1.1 Catalog: MarkInfo and StructTreeRoot

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

```
--tag/struct/0 This is the object for the root object, the StructTreeRoot
20 \pdf_object_new:nn { __tag/struct/0 }{ dict }
(End definition for __tag/struct/0.)
21 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}
22 {
23   \bool_if:NT \g__tag_active_tree_bool
24   {
25     \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
26     \pdfmanagement_add:nnx
```

```

27         { Catalog }
28         { StructTreeRoot }
29         { \pdf_object_ref:n { __tag/struct/0 } }
30     }
31 }

```

1.2 Writing structure elements

The following commands are needed to write out the structure.

`__tag_tree_write_structtreeroot:` This writes out the root object.

```

32 \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
33 {
34     \__tag_prop_gput:cnx
35     { g__tag_struct_0_prop }
36     { ParentTree }
37     { \pdf_object_ref:n { __tag/tree/parenttree } }
38     \__tag_prop_gput:cnx
39     { g__tag_struct_0_prop }
40     { RoleMap }
41     { \pdf_object_ref:n { __tag/tree/rolemap } }
42     \__tag_struct_write_obj:n { 0 }
43 }

```

(End definition for __tag_tree_write_structtreeroot:.)

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

```

44 \cs_new_protected:Npn \__tag_tree_write_structelements:
45 {
46     \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
47     {
48         \__tag_struct_write_obj:n { ##1 }
49     }
50 }

```

(End definition for __tag_tree_write_structelements:.)

1.3 ParentTree

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

```

51 \pdf_object_new:nn { __tag/tree/parenttree }{ dict }

```

(End definition for __tag/tree/parenttree.)

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

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

```

52 \newcounter { g__tag_parenttree_obj_int }
53 \hook_gput_code:nnn{begindocument}{tagpdf}
54 {

```

```

55 \int_gset:Nn
56 \c@g__tag_parenttree_obj_int
57 { \__tag_ref_value_lastpage:nn{abspage}{100} }
58 }

```

(End definition for \c@g__tag_parenttree_obj_int.)

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

\g__tag_parenttree_objr_tl

```

59 \tl_new:N \g__tag_parenttree_objr_tl

```

(End definition for \g__tag_parenttree_objr_tl.)

__tag_parenttree_add_objr:nn

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

```

60 \cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 % #1 StructParent number, #2 objref
61 {
62   \tl_gput_right:Nx \g__tag_parenttree_objr_tl
63   {
64     #1 \c_space_tl #2 ^^J
65   }
66 }

```

(End definition for __tag_parenttree_add_objr:nn.)

\l__tag_parenttree_content_tl

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

```

67 \tl_new:N \l__tag_parenttree_content_tl

```

(End definition for \l__tag_parenttree_content_tl.)

__tag_tree_fill_parenttree:

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

```

68
69 \cs_new_protected:Npn \__tag_tree_fill_parenttree:
70 {
71   \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{abspage}{-1}} %not quite clear i
72   { %page ##1
73     \prop_clear:N \l__tag_tmpa_prop
74     \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{tagmcabs}{-1}}
75     {
76       %mcid###1
77       \int_compare:nT
78       {\__tag_ref_value:enn{mcid-###1}{tagabspage}{-1}=##1} %mcid is on current page
79       {% yes
80         \prop_put:Nxx
81         \l__tag_tmpa_prop
82         {\__tag_ref_value:enn{mcid-###1}{tagmcid}{-1}}
83         {\prop_item:Nn \g__tag_mc_parenttree_prop {###1}}
84       }
85     }
86     \tl_put_right:Nx\l__tag_parenttree_content_tl
87     {
88       \int_eval:n {##1-1}\c_space_tl
89       [\c_space_tl %]

```

```

90     }
91     \int_step_inline:nnnn
92     {0}
93     {1}
94     { \prop_count:N \l__tag_tmpa_prop -1 }
95     {
96         \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
97         {% page#1:mcid##1:\l__tag_tmpa_tl :content
98         \tl_put_right:Nx \l__tag_parenttree_content_tl
99         {
100             \pdf_object_if_exist:eT { __tag/struct/\l__tag_tmpa_tl }
101             {
102                 \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
103             }
104             \c_space_tl
105         }
106     }
107     {
108         \msg_warning:nn { tag } {tree-mcid-index-wrong}
109     }
110 }
111 \tl_put_right:Nn
112 \l__tag_parenttree_content_tl
113 {%[
114 ]^^J
115 }
116 }
117 }

```

(End definition for __tag_tree_fill_parenttree:.)

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

```

118 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
119 {
120     \tl_set:Nn \l__tag_parenttree_content_tl
121     {
122         \lua_now:e
123         {
124             ltx.__tag.func.output_parenttree
125             (
126                 \int_use:N\g_shipout_readonly_int
127             )
128         }
129     }
130 }

```

(End definition for __tag_tree_lua_fill_parenttree:.)

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

```

131 \cs_new_protected:Npn \__tag_tree_write_parenttree:
132 {
133     \bool_if:NTF \g__tag_mode_lua_bool
134     {

```

```

135     \__tag_tree_lua_fill_parenttree:
136   }
137   {
138     \__tag_tree_fill_parenttree:
139   }
140   \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
141   \pdf_object_write:nx { __tag/tree/parenttree }
142   {
143     /Nums\c_space_tl [\l__tag_parenttree_content_tl]
144   }
145 }

```

(End definition for __tag_tree_write_parenttree:.)

1.4 Rolemap dictionary

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

```

__tag/tree/rolemap At first we reserve again an object.
146 \pdf_object_new:nn { __tag/tree/rolemap }{ dict }

```

(End definition for __tag/tree/rolemap.)

```

\__tag_tree_write_rolemap: This writes out the rolemap, basically it simply pushes out the dictionary which has been
filled in the role module.
147 \cs_new_protected:Npn \__tag_tree_write_rolemap:
148   {
149     \pdf_object_write:nx { __tag/tree/rolemap }
150     {
151       \pdfdict_use:n{g__tag_role/RoleMap_dict}
152     }
153   }

```

(End definition for __tag_tree_write_rolemap:.)

1.5 Classmap dictionary

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

```

\__tag_tree_write_classmap:
154 \cs_new_protected:Npn \__tag_tree_write_classmap:
155   {
156     \tl_clear:N \l__tag_tmpa_tl
157     \seq_gremove_duplicates:N \g__tag_attr_class_used_seq
158     \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
159     {
160       /##1\c_space_tl
161       <<
162       \prop_item:Nn
163       \g__tag_attr_entries_prop
164       {##1}

```

```

165         >>
166     }
167     \tl_set:Nx \l__tag_tmpa_tl
168     {
169         \seq_use:Nn
170         \l__tag_tmpa_seq
171         { \iow_newline: }
172     }
173     \tl_if_empty:NF
174     \l__tag_tmpa_tl
175     {
176         \pdf_object_new:nn { __tag/tree/classmap }{ dict }
177         \pdf_object_write:nx
178         { __tag/tree/classmap }
179         { \l__tag_tmpa_tl }
180         \__tag_prop_gput:cnx
181         { g__tag_struct_0_prop }
182         { ClassMap }
183         { \pdf_object_ref:n { __tag/tree/classmap } }
184     }
185 }

```

(End definition for __tag_tree_write_classmap:.)

1.6 Namespaces

Namespaces are handle in the role module, here is the code to write them out. Namespaces are only relevant for pdf2.0 but we don't care, it doesn't harm.

__tag/tree/namespaces

```

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

```

(End definition for __tag/tree/namespaces.)

__tag_tree_write_namespaces:

```

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

```

```

207     \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
208   }
209 }

```

(End definition for _tag_tree_write_namespaces:.)

1.7 Finishing the structure

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

_tag_finish_structure:

```

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

```

(End definition for _tag_finish_structure:.)

1.8 StructParents entry for Page

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

```

223 \hook_gput_code:nnn{begindocument}{tagpdf}
224 {
225   \bool_if:NT\g__tag_active_tree_bool
226   {
227     \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
228     {
229       \pdfmanagement_add:nnx
230       { Page }
231       { StructParents }
232       { \int_eval:n { \g_shipout_readonly_int } }
233     }
234   }
235 }
236 </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 alttext for now.

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

2 Public keys

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

<u>tag</u>	This key is required, unless <code>artifact</code> is used. The value is a tag like <code>P</code> or <code>H1</code> 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 <code>H4</code> is fine).
<u>artifact</u>	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 <code>pagination</code> , <code>layout</code> , <code>page</code> , <code>background</code> and <code>notype</code> (this is the default).
<u>raw</u>	This key allows to add more entries to the properties dictionary. The value must be correct, low-level PDF. E.g. <code>raw=/Alt (Hello)</code> will insert an alternative Text.
<u>alttext</u> <u>alttext-o</u>	This key inserts an <code>/Alt</code> value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. With <code>alttext-o</code> the value is expanded once.
<u>actualtext</u> <u>actualtext-o</u>	This key inserts an <code>/ActualText</code> value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. With <code>actualtext-o</code> the value is expanded once.
<u>label</u>	This key sets a label by which one can call the marked content later in another structure (if it has been stashed with the <code>stash</code> key). Internally the label name will start with <code>tagpdf-</code> .
<u>stash</u>	<p>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.</p> <p>The code is splitted into three parts: code shared by all engines, code specific to <code>luamode</code> and code not used by <code>luamode</code>.</p>

3 Marked content code – shared

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-mc-code-shared} {2021-07-03} {0.91}
4 {part of tagpdf - code related to marking chunks -
5   code shared by generic and luamode }
6 </header>
```

3.1 Variables and counters

MC chunks must be counted. I use a latex counter for the absolute count, so that it is added to `\cl@@ckpt` and restored e.g. in `tabulars` and `align`. `\int_new:N \c@g_@@_MCID_int` and `\tl_put_right:Nn\cl@@ckpt{\@elt{g_uf_test_int}}` would work too, but as the name is not `expl3` then too, why bother? The absolute counter can be used to label and to check if the page counter needs a reset.

`g__tag_MCID_abs_int`

```
7 \{*shared}
8 \newcounter { g__tag_MCID_abs_int }
```

(End definition for `g__tag_MCID_abs_int`.)

`__tag_get_mc_abs_cnt:` A (expandable) function to get the current value of the cnt.

```
9 \cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }
```

(End definition for `__tag_get_mc_abs_cnt:`.)

`\g__tag_MCID_tmp_bypage_int`

The following hold the temporary by page number assigned to a mc. It must be defined in the shared code to avoid problems with labels.

```
10 \int_new:N \g__tag_MCID_tmp_bypage_int
```

(End definition for `\g__tag_MCID_tmp_bypage_int`.)

`\g__tag_in_mc_bool`

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

```
11 \bool_new:N \g__tag_in_mc_bool
```

(End definition for `\g__tag_in_mc_bool`.)

`\g__tag_mc_parenttree_prop`

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

key: absolute number of the mc (`tagmcabs`)

value: the structure number the mc is in

```
12 \__tag_prop_new:N \g__tag_mc_parenttree_prop
```

(End definition for `\g__tag_mc_parenttree_prop`.)

`\g__tag_mc_parenttree_prop`

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

```
13 \seq_new:N \g__tag_mc_stack_seq
```

(End definition for `\g__tag_mc_parenttree_prop`.)

`\l__tag_mc_artifact_type_tl`

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

```
14 \tl_new:N \l__tag_mc_artifact_type_tl
```

(End definition for `\l__tag_mc_artifact_type_tl`.)

`\l__tag_mc_key_stash_bool`

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

`\l__tag_mc_artifact_bool`

```
15 \bool_new:N \l__tag_mc_key_stash_bool
```

```
16 \bool_new:N \l__tag_mc_artifact_bool
```

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

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

`\g__tag_mc_key_tag_tl`

`\l__tag_mc_key_label_tl` 17 \tl_new:N \l__tag_mc_key_tag_tl

`\l__tag_mc_key_properties_tl` 18 \tl_new:N \g__tag_mc_key_tag_tl

19 \tl_new:N \l__tag_mc_key_label_tl

20 \tl_new:N \l__tag_mc_key_properties_tl

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

3.2 Functions

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

`tagabspace`: the absolute page, `\g_shipout_readonly_int`,

`tagmcabs`: the absolute mc-counter `\c@g_@@MCID_abs_int`,

`tagmcid`: the ID of the chunk on the page `\g_@@MCID_tmp_bypage_int`, this typically settles down after a second compilation. The reference command is defined in `tagpdf.dtx` and is based on `l3ref`.

21 \cs_new:Nn __tag_mc_handle_mc_label:n

22 {

23 __tag_ref_label:en{tagpdf-#1}{mc}

24 }

(End definition for `__tag_mc_handle_mc_label:n`.)

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

25 \cs_new_protected:Npn __tag_mc_set_label_used:n #1 %#1 labelname

26 {

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

28 }

(End definition for `__tag_mc_set_label_used:n`.)

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

TODO: is testing for struct the right test?

29 \cs_new_protected:Npn \tag_mc_use:n #1 %#1: label name

30 {

31 __tag_check_if_active_struct:T

32 {

33 \tl_set:Nx \l__tag_tmpa_tl { __tag_ref_value:nnn{tagpdf-#1}{tagmcabs}{ } }

34 \tl_if_empty:NTF\l__tag_tmpa_tl

35 {

36 \msg_warning:nnn {tag} {mc-label-unknown} {#1}

37 }

38 {

39 \cs_if_free:cTF { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }

40 {

41 __tag_mc_handle_stash:x { \l__tag_tmpa_tl }

```

42         \__tag_mc_set_label_used:n {#1}
43     }
44     {
45         \msg_warning:nnn {tag}{mc-used-twice}{#1}
46     }
47 }
48 }
49 }

```

(End definition for `\tag_mc_use:n`. This function is documented on page 41.)

`\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.

```

50 \cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1
51 {
52     \tag_mc_end_push:
53     \tag_mc_begin:n {artifact=#1}
54     \tag_stop_group_begin:
55 }
56
57 \cs_new_protected:Npn \tag_mc_artifact_group_end:
58 {
59     \tag_stop_group_end:
60     \tag_mc_end:
61     \tag_mc_begin_pop:n{ }
62 }

```

(End definition for `\tag_mc_artifact_group_begin:n` and `\tag_mc_artifact_group_end:`. These functions are documented on page 41.)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

```

63 \cs_new_protected:Npn \tag_mc_end_push:
64 {
65     \__tag_check_if_active_mc:T
66     {
67         \__tag_mc_if_in:TF
68         {
69             \seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
70             \__tag_check_mc_pushed_popped:nn
71             { pushed }
72             { \tag_get:n {mc_tag} }
73             \tag_mc_end:
74         }
75         {
76             \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
77             \__tag_check_mc_pushed_popped:nn { pushed }{-1}
78         }
79     }
80 }
81
82 \cs_new_protected:Npn \tag_mc_begin_pop:n #1
83 {
84     \__tag_check_if_active_mc:T
85     {
86         \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl

```

```

87     {
88         \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
89         {
90             \__tag_check_mc_pushed_popped:nn {popped}{-1}
91         }
92         {
93             \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
94             \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
95         }
96     }
97     {
98         \__tag_check_mc_pushed_popped:nn {popped}{empty-stack,~nothing}
99     }
100 }
101 }

```

(End definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 41.)

3.3 Keys

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

```

stash the two internal artifact keys are use to define the public artifact.
__artifact-bool 102 \keys_define:nn { __tag / mc }
__artifact-type 103 {
104     stash .bool_set:N = \l__tag_mc_key_stash_bool,
105     __artifact-bool .bool_set:N = \l__tag_mc_artifact_bool,
106     __artifact-type .choice:,
107     __artifact-type / pagination .code:n =
108     {
109         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
110     },
111     __artifact-type / layout .code:n =
112     {
113         \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
114     },
115     __artifact-type / page .code:n =
116     {
117         \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
118     },
119     __artifact-type / background .code:n =
120     {
121         \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
122     },
123     __artifact-type / notype .code:n =
124     {
125         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
126     },
127     __artifact-type / .code:n =
128     {
129         \tl_set:Nn \l__tag_mc_artifact_type_tl {}
130     },
131 }

```

(End definition for `stash`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 63.)

132 `</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} {2021-07-03} {0.91}
4 {part of tagpdf - code related to marking chunks - generic mode}
5 </generic>
```

1.1 Variables

`\g__tag_MCID_byabspage_prop` This property will hold the current maximum on a page it will contain key-value of type $\langle abspagenum \rangle = \langle max\ mcid \rangle$

```
6 <*generic>
7 \__tag_prop_new:N \g__tag_MCID_byabspage_prop
(End definition for \g__tag_MCID_byabspage_prop.)
```

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

```
8 \tl_new:N \l__tag_mc_ref_abspage_tl
(End definition for \l__tag_mc_ref_abspage_tl.)
```

`\l__tag_mc_tmpa_tl` temporary variable

```
9 \tl_new:N \l__tag_mc_tmpa_tl
(End definition for \l__tag_mc_tmpa_tl.)
```

1.2 Functions

`__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
10 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
11 {
12   \bool_if:NTF \g__tag_in_mc_bool
13   { \prg_return_true: }
14   { \prg_return_false: }
15 }
16
17 \prg_new_eq_conditional:Nnn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}
```

(End definition for `__tag_mc_if_in:TF` and `\tag_mc_if_in:TF`. This function is documented on page 41.)

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

```

18 % #1 tag, #2 properties
19 \cs_set_eq:NN \__tag_mc_bmc:n \pdf_bmc:n
20 \cs_set_eq:NN \__tag_mc_emc: \pdf_emc:
21 \cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
22 \cs_generate_variant:Nn \__tag_mc_bdc:nn {nx}

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

```

`__tag_mc_bdc_mcid:nn` This create a BDC mark with an /MCID key. Most of the work here is to get the current
`__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. We also define a wrapper around the low-level command as luamode
will need something different.

```

23 \cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
24 {
25   \int_gincr:N \c@g__tag_MCID_abs_int
26   \tl_set:Nx \l__tag_mc_ref_abspage_tl
27   {
28     \__tag_ref_value:enn %3 args
29     {
30       mcid-\int_use:N \c@g__tag_MCID_abs_int
31     }
32     { tagabspage }
33     {-1}
34   }
35   \prop_get:NoNTF
36   \g__tag_MCID_byabspage_prop
37   {
38     \l__tag_mc_ref_abspage_tl
39   }
40   \l__tag_mc_tmpa_tl
41   {
42     %key already present, use value for MCID and add 1 for the next
43     \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
44     \__tag_prop_gput:Nxx
45     \g__tag_MCID_byabspage_prop
46     { \l__tag_mc_ref_abspage_tl }
47     { \int_eval:n {\l__tag_mc_tmpa_tl +1} }
48   }
49   {
50     %key not present, set MCID to 0 and insert 1
51     \int_gzero:N \g__tag_MCID_tmp_bypage_int
52     \__tag_prop_gput:Nxx
53     \g__tag_MCID_byabspage_prop
54     { \l__tag_mc_ref_abspage_tl }
55     {1}
56   }
57   \__tag_ref_label:en
58   {

```

```

59     mcid-\int_use:N \c@g__tag_MCID_abs_int
60   }
61   { mc }
62   \__tag_mc_bdc:nx
63   {#1}
64   { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
65 }
66 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
67 {
68   \__tag_mc_bdc_mcid:nn {#1} {}
69 }
70
71 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %1 tag, #2 properties
72 {
73   \__tag_mc_bdc_mcid:nn {#1} {#2}
74 }
75
76 \cs_generate_variant:Nn \__tag_mc_handle_mcid:nn {VV}

```

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

__tag_mc_handle_stash:n
__tag_mc_handle_stash:x

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?

```

77 \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
78 {
79   \__tag_check_mc_used:n {#1}
80   \__tag_struct_kid_mc_gput_right:nn
81   { \g__tag_struct_stack_current_tl }
82   {#1}
83   \prop_gput:Nxx \g__tag_mc_parenttree_prop
84   {#1}
85   { \g__tag_struct_stack_current_tl }
86 }
87 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { x }

```

(End definition for __tag_mc_handle_stash:n.)

__tag_mc_bmc_artifact:
__tag_mc_bmc_artifact:n
__tag_mc_handle_artifact:N

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

```

88 \cs_new_protected:Npn \__tag_mc_bmc_artifact:
89 {
90   \__tag_mc_bmc:n {Artifact}
91 }
92 \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
93 {
94   \__tag_mc_bdc:nn {Artifact}{/Type/#1}
95 }
96 \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
97   % #1 is a var containing the artifact type
98 {

```

```

99     \tl_if_empty:NTF #1
100     { \__tag_mc_bmc_artifact: }
101     { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
102 }

(End definition for \__tag_mc_bmc_artifact:, \__tag_mc_bmc_artifact:n, and \__tag_mc_handle_
artifact:N.)

```

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

```

103 \cs_new:Nn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }

(End definition for \__tag_get_data_mc_tag:.)

```

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

\tag_mc_end:

```

104 \cs_new_protected:Npn \tag_mc_begin:n #1 %#1 keyval
105 {
106     \__tag_check_if_active_mc:T
107     {
108         \group_begin: %hm
109         \__tag_check_mc_if_nested:
110         \bool_gset_true:N \g__tag_in_mc_bool
111         \keys_set:nn { \__tag / mc } {#1}
112         \bool_if:NTF \l__tag_mc_artifact_bool
113         { %handle artifact
114             \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
115         }
116         { %handle mcid type
117             \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
118             \__tag_mc_handle_mcid:VV
119             \l__tag_mc_key_tag_tl
120             \l__tag_mc_key_properties_tl
121             \tl_if_empty:NF {\l__tag_mc_key_label_tl}
122             {
123                 \exp_args:NV
124                 \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
125             }
126             \bool_if:NF \l__tag_mc_key_stash_bool
127             {
128                 \__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
129             }
130         }
131     } \group_end:
132 }
133 }
134 \cs_new_protected:Nn \tag_mc_end:
135 {
136     \__tag_check_if_active_mc:T
137     {
138         \__tag_check_mc_if_open:
139         \bool_gset_false:N \g__tag_in_mc_bool
140         \tl_gset:Nn \g__tag_mc_key_tag_tl { }

```

```

141     \_tag_mc_emc:
142     }
143 }

```

(End definition for `\tag_mc_begin:n` and `\tag_mc_end:.` These functions are documented on page 41.)

1.3 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
raw
alttext
alttext-o
actualtext
actualtext-o
label
artifact
144 \keys_define:nn { _tag / mc }
145 {
146   tag .code:n = % the name (H,P,Span) etc
147   {
148     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
149     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
150   },
151   raw .code:n =
152   {
153     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
154   },
155   alttext .code:n = % Alt property
156   {
157     \str_set_convert:Nnon
158       \l__tag_tmpa_str
159       { #1 }
160       { default }
161       { utf16/hex }
162     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
163     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
164   },
165   alttext-o .code:n = % Alt property
166   {
167     \str_set_convert:Nnon
168       \l__tag_tmpa_str
169       { #1 }
170       { default }
171       { utf16/hex }
172     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
173     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
174   },
175   actualtext .code:n = % ActualText property
176   {
177     \str_set_convert:Nnon
178       \l__tag_tmpa_str
179       { #1 }
180       { default }
181       { utf16/hex }
182     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
183     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
184   },
185   actualtext-o .code:n = % ActualText property

```

```

186     {
187         \str_set_convert:Noon
188         \l__tag_tmpa_str
189         { #1 }
190         { default }
191         { utf16/hex }
192         \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
193         \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
194     },
195     label .tl_set:N      = \l__tag_mc_key_label_tl,
196     artifact .code:n    =
197     {
198         \exp_args:Nnx
199         \keys_set:nn
200         { __tag / mc }
201         { __artifact-bool, __artifact-type=#1 }
202     },
203     artifact .default:n = {notype}
204 }
205 </generic>

```

(End definition for tag and others. These functions are documented on page 62.)

Part VI

The **tagpdf-mc-luacode** module

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

Part of the tagpdf package

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

1 Marked content code – luamode code

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

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

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

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

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

`tag` : the type (a string)

`raw` : more properties (string)

`label`: a string.

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

`kids`: a array of tables

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

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

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

```
1 <@@=tag>
2 <*luamode>
3 \ProvidesExplPackage {tagpdf-mc-code-lua} {2021-07-03} {0.91}
4   {tagpdf - mc code only for the luamode }
5 </luamode>
```

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

```
6 <*luamode>
7 \hook_gput_code:nnn{begindocument}{tagpdf/mc}
8 {
```

```

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

```

1.1 Commands

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

\tag_mc_if_in: 58 \prg_new_conditional:Nnn __tag_mc_if_in: {p,T,F,TF}

```

59 {
60   \int_compare:nNnTF
61     { -2147483647 }
62     =
63     {\lua_now:e
64       {
65         tex.print(tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr))
66       }
67     }
68     { \prg_return_false: }
69     { \prg_return_true: }
70   }
71
72   \prg_new_eq_conditional:NNn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}

```

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

```

\__tag_mc_lua_set_mc_type_attr:n This takes a tag name, and sets the attributes to the related number. It is not decided
\__tag_mc_lua_set_mc_type_attr:o yet if this will be global or local, see the global-mc option.
\__tag_mc_lua_unset_mc_type_attr:
73 \cs_new:Nn \__tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
74 {
75   %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
76   \tl_set:Nx\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")}} }
77   \lua_now:e
78   {
79     tex.setattribute
80     (
81       "global",
82       luatexbase.attributes.g__tag_mc_type_attr,
83       \l__tag_tmpa_tl
84     )
85   }
86   \lua_now:e
87   {
88     tex.setattribute
89     (
90       "global",
91       luatexbase.attributes.g__tag_mc_cnt_attr,
92       \__tag_get_mc_abs_cnt:
93     )
94   }
95 }
96
97 \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
98
99 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
100 {
101   \lua_now:e
102   {
103     tex.setattribute
104     (
105       "global",
106       luatexbase.attributes.g__tag_mc_type_attr,
107       -2147483647

```



```

108     )
109   }
110   \lua_now:e
111   {
112     tex.setattribute
113     (
114       "global",
115       luatexbase.attributes.g__tag_mc_cnt_attr,
116       -2147483647
117     )
118   }
119 }
120

```

(End definition for _tag_mc_lua_set_mc_type_attr:n and _tag_mc_lua_unset_mc_type_attr:.)

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

```

121 \cs_new:Nn \_tag_mc_insert_mcid_kids:n
122 {
123   \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
124 }
125
126 \cs_new:Nn \_tag_mc_insert_mcid_single_kids:n
127 {
128   \lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
129 }

```

(End definition for _tag_mc_insert_mcid_kids:n and _tag_mc_insert_mcid_single_kids:n.)

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

```

130 \cs_new:Nn \_tag_mc_handle_stash:n %1 mcidnum
131 {
132   \_tag_check_mc_used:n { #1 }
133   \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
134                     % so use the kernel command
135   { g__tag_struct_kids \g__tag_struct_stack_current_tl _seq }
136   {
137     \_tag_mc_insert_mcid_kids:n {#1}%
138   }
139   \lua_now:e
140   {
141     ltx.__tag.func.store_struct_mcabs
142     (
143       \g__tag_struct_stack_current_tl,#1
144     )
145   }
146   \prop_gput:Nxx
147   \g__tag_mc_parenttree_prop
148   { #1 }
149   { \g__tag_struct_stack_current_tl }
150 }
151
152 \cs_generate_variant:Nn \_tag_mc_handle_stash:n { x }

```

(End definition for `_tag_mc_handle_stash:n`.)

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

```

153 \cs_new_protected:Nn \tag_mc_begin:n
154 {
155   \__tag_check_if_active_mc:T
156   {
157     \group_begin:
158     %\__tag_check_mc_if_nested:
159     \bool_gset_true:N \g__tag_in_mc_bool
160     \bool_set_false:N \l__tag_mc_artifact_bool
161     \tl_clear:N \l__tag_mc_key_properties_tl
162     \int_gincr:N \c@g__tag_MCID_abs_int
163     \keys_set:nn { __tag / mc } { label={}, #1 }
164     %check that a tag or artifact has been used
165     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
166     %set the attributes:
167     \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
168     \bool_if:NF \l__tag_mc_artifact_bool
169     { % store the absolute num name in a label:
170       \tl_if_empty:NF {\l__tag_mc_key_label_tl}
171       {
172         \exp_args:NV
173         \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
174       }
175       % if not stashed record the absolute number
176       \bool_if:NF \l__tag_mc_key_stash_bool
177       {
178         \__tag_mc_handle_stash:x { \__tag_get_mc_abs_cnt: }
179       }
180     }
181     \group_end:
182   }
183 }

```

(End definition for `\tag_mc_begin:n`. This function is documented on page 41.)

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

```

184 \cs_new_protected:Nn \tag_mc_end:
185 {
186   \__tag_check_if_active_mc:T
187   {
188     %\__tag_check_mc_if_open:
189     \bool_gset_false:N \g__tag_in_mc_bool
190     \bool_set_false:N \l__tag_mc_artifact_bool
191     \__tag_mc_lua_unset_mc_type_attr:
192     \tl_set:Nn \l__tag_mc_key_tag_tl { }
193     \tl_gset:Nn \g__tag_mc_key_tag_tl { }
194   }
195 }

```

(End definition for `\tag_mc_end:`. This function is documented on page 41.)

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

```
196 \cs_new:Npn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
(End definition for \__tag_get_data_mc_tag:.)
```

1.2 Key definitions

```

tag      TODO: check conversion, check if local/global setting is right.
raw      197 \keys_define:nn { __tag / mc }
alttext  198 {
alttext-o 199   tag .code:n = %
actualtext 200   {
actualtext-o 201     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
label      202     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
artifact   203     \lua_now:e
204     {
205       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
206     }
207   },
208   raw .code:n =
209   {
210     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
211     \lua_now:e
212     {
213       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
214     }
215   },
216   alttext .code:n      = % Alt property
217   {
218     \str_set_convert:Nnon
219     \l__tag_tmpa_str
220     { #1 }
221     { default }
222     { utf16/hex }
223     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
224     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
225     \lua_now:e
226     {
227       ltx.__tag.func.store_mc_data
228       (
229         \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
230       )
231     }
232   },
233   alttext-o .code:n      = % Alt property
234   {
235     \str_set_convert:Nnon
236     \l__tag_tmpa_str
237     { #1 }
238     { default }
239     { utf16/hex }
240     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
241     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }

```

```

242     \lua_now:e
243     {
244         ltx.__tag.func.store_mc_data
245         (
246             \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
247         )
248     }
249 },
250 actualtext .code:n      = % Alt property
251 {
252     \str_set_convert:Nnon
253     \l__tag_tmpa_str
254     { #1 }
255     { default }
256     { utf16/hex }
257     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
258     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
259     \lua_now:e
260     {
261         ltx.__tag.func.store_mc_data
262         (
263             \__tag_get_mc_abs_cnt:,"actualtext","/ActualText~<\str_use:N \l__tag_tmpa_str>"
264         )
265     }
266 },
267 actualtext-o .code:n    = % Alt property
268 {
269     \str_set_convert:Noon
270     \l__tag_tmpa_str
271     { #1 }
272     { default }
273     { utf16/hex }
274     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
275     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
276     \lua_now:e
277     {
278         ltx.__tag.func.store_mc_data
279         (
280             \__tag_get_mc_abs_cnt:,
281             "actualtext",
282             "/ActualText~<\str_use:N \l__tag_tmpa_str>"
283         )
284     }
285 },
286 label .code:n =
287 {
288     \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
289     \lua_now:e
290     {
291         ltx.__tag.func.store_mc_data
292         (
293             \__tag_get_mc_abs_cnt:,"label","#1"
294         )
295     }

```

```

296     },
297     __artifact-store .code:n =
298     {
299         \lua_now:e
300         {
301             ltx.__tag.func.store_mc_data
302             (
303                 \__tag_get_mc_abs_cnt:,"artifact","#1"
304             )
305         }
306     },
307     artifact .code:n      =
308     {
309         \exp_args:Nnx
310         \keys_set:nn
311         { __tag / mc }
312         { __artifact-bool, __artifact-type=#1, tag=Artifact }
313         \exp_args:Nnx
314         \keys_set:nn
315         { __tag / mc }
316         { __artifact-store=\l__tag_mc_artifact_type_tl }
317     },
318     artifact .default:n    = { notype }
319 }
320
321 </luamode>

```

(End definition for tag and others. These functions are documented on page 62.)

Part VII

The tagpdf-struct module

Commands to create the structure

Part of the tagpdf package

1 Public Commands

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

These commands start and end a new structure. They don't start a group. They set all their values globally.

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

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

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

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

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

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

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

2 Public keys

2.1 Keys for the structure commands

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

<u>stash</u>	Normally a new structure inserts itself as a kid into the currently active structure. This key prohibits this. The structure is nevertheless from now on “the current active structure” and parent for following marked content and structures.
<u>label</u>	This key sets a label by which one can use the structure later in another structure. Internally the label name will start with <code>tagpdfstruct-</code> .
<u>title</u> <u>title-o</u>	This keys allows to set the dictionary entry <code>/Title</code> in the structure object. The value is handled as verbatim string and hex encoded. Commands are not expanded. <code>title-o</code> will expand the value once.
<u>alttext</u> <u>alttext-o</u>	This key inserts an <code>/Alt</code> value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. <code>alttext-o</code> will expand the value once.
<u>actualtext</u> <u>actualtext-o</u>	This key inserts an <code>/ActualText</code> value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. <code>actualtext-o</code> will expand the value once.
<u>lang</u>	This key allows to set the language for a structure element. The value should be a bcp-identifier, e.g. <code>de-De</code> .
<u>ref</u>	This key allows to add references to other structure elements, it adds the <code>/Ref</code> array to the structure. The value should be a comma separated list of structure labels set with the <code>label</code> key. e.g. <code>ref={label1,label2}</code> .
<u>E</u>	This key sets the <code>/E</code> key, the expanded form of an abbreviation or an acronym (I couldn’t think of a better name, so I stucked to E).
<u>AF</u> <u>AFinline</u> <u>AFinline-o</u>	<p>AF = <code><object name></code> AFinline = <code><text content></code></p> <p>These keys allows to reference an associated file in the structure element. The value <code><object name></code> should be the name of an object pointing to the <code>/Filespec</code> dictionary as expected by <code>\pdf_object_ref:n</code> from a current <code>l3kernel</code>.</p> <p>The value <code>AF-inline</code> is some text, which is embedded in the PDF as a text file with mime type <code>text/plain</code>. <code>AF-inline-o</code> is like <code>AF-inline</code> but expands the value once.</p> <p>Future versions will perhaps extend this to more mime types, but it is still a research task to find out what is really needed.</p>

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

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

2.2 Setup keys

newattribute `newattribute = {\langle name \rangle}{\langle Content \rangle}`

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

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

1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-struct-code} {2021-07-03} {0.91}
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 <*package>
7 \newcounter { g__tag_struct_abs_int }
8 \int_gzero:N \c@g__tag_struct_abs_int
```

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

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

```
9 \__tag_seq_new:N \g__tag_struct_objR_seq
```


(End definition for \g__tag_struct_objR_seq.)

\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.

```

10 \seq_new:N      \g__tag_struct_stack_seq
11 \seq_gpush:Nn   \g__tag_struct_stack_seq {0}

```

(End definition for \g__tag_struct_stack_seq.)

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

```

12 \seq_new:N      \g__tag_struct_tag_stack_seq
13 \seq_gpush:Nn   \g__tag_struct_tag_stack_seq {Root}

```

(End definition for \g__tag_struct_tag_stack_seq.)

\g__tag_struct_stack_current_tl The global variable will hold the current structure number. The local temporary variable will hold the parent when we fetch it from the stack.

```

14 \tl_new:N      \g__tag_struct_stack_current_tl
15 \tl_new:N      \l__tag_struct_stack_parent_tmpa_tl

```

(End definition for \g__tag_struct_stack_current_tl and \l__tag_struct_stack_parent_tmpa_tl.)

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

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

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

All properties have at least the keys

Type StructTreeRoot or StructElem

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

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

```

16 \seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
17   {%p. 857/858
18     Type,                % always /StructTreeRoot
19     K,                   % kid, dictionary or array of dictionaries
20     IDTree,              % currently unused
21     ParentTree,          % required,obj ref to the parent tree
22     ParentTreeNextKey,   % optional
23     RoleMap,
24     ClassMap,
25     Namespaces
26   }
27
28 \seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
29   {%p 858 f
30     Type,                %always /StructElem

```

```

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

```

(End definition for \c__tag_struct_StructTreeRoot_entries_seq and \c__tag_struct_StructElem_entries_seq.)

3.1 Variables used by the keys

\g__tag_struct_tag_tl Use by the tag key to store the tag and the namespace.

```

\g__tag_struct_tag_NS_tl 52 \tl_new:N \g__tag_struct_tag_tl
53 \tl_new:N \g__tag_struct_tag_NS_tl

```

(End definition for \g__tag_struct_tag_tl and \g__tag_struct_tag_NS_tl.)

\l__tag_struct_key_label_tl This will hold the label value.

```

54 \tl_new:N \l__tag_struct_key_label_tl

```

(End definition for \l__tag_struct_key_label_tl.)

\l__tag_struct_elem_stash_bool This will keep track of the stash status

```

55 \bool_new:N \l__tag_struct_elem_stash_bool

```

(End definition for \l__tag_struct_elem_stash_bool.)

4 Commands

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

```

__tag_struct_output_prop_aux:nn
__tag_new_output_prop_handler:n
56 \cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
57 {
58   \prop_if_in:cnT
59     { g__tag_struct_#1_prop }
60     { #2 }
61     {
62       \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
63     }
64 }
65
66 \cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
67 {
68   \cs_new:cn { __tag_struct_output_prop_#1:n }
69   {
70     \__tag_struct_output_prop_aux:nn {#1}{##1}
71   }
72 }

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

```

4.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object is @@/struct/0 which is currently created in the tree code (TODO move it here). The ParentTree and RoleMap entries are added at begin document in the tree code as they refer to object which are setup in other parts of the code. This avoid timing issues.

```

73 \tl_gset:Nn \g__tag_struct_stack_current_tl {0}

g__tag_struct_0_prop
g__tag_struct_kids_0_seq
74 \__tag_prop_new:c { g__tag_struct_0_prop }
75 \__tag_new_output_prop_handler:n {0}
76 \__tag_seq_new:c { g__tag_struct_kids_0_seq }
77
78 \__tag_prop_gput:cnn
79 { g__tag_struct_0_prop }
80 { Type }
81 { /StructTreeRoot }
82
83
84

```

Namespaces are pdf 2.0 but it doesn't harm to have an empty entry. We could add a test, but if the code moves into the kernel, timing could get tricky.

```

85 \__tag_prop_gput:cnx
86 { g__tag_struct_0_prop }
87 { Namespaces }
88 { \pdf_object_ref:n { __tag/tree/namespaces } }

(End definition for g__tag_struct_0_prop and g__tag_struct_kids_0_seq.)

```

4.2 Handlings kids

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

`_tag_struct_kid_mc_gput_right:nn`

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.

```

89 \cs_new_protected:Npn \_tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 MCID abs
90 {
91   \_tag_seq_gput_right:cx
92   { g\_tag_struct_kids_#1_seq }
93   {
94     <<
95     /Type \c_space_tl /MCR \c_space_tl
96     /Pg
97     \c_space_tl
98     \pdf_pageobject_ref:n { \_tag_ref_value:enn{mcid-#2}{tagabspage}{1} }
99     /MCID \c_space_tl \_tag_ref_value:enn{mcid-#2}{tagmcid}{1}
100    >>
101   }
102 }
```

(End definition for `_tag_struct_kid_mc_gput_right:nn`.)

`_tag_struct_kid_struct_gput_right:nn`

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

`_tag_struct_kid_struct_gput_right:xx`

```

103 \cs_new_protected:Npn \_tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent struct, #
104 {
105   \_tag_seq_gput_right:cx
106   { g\_tag_struct_kids_#1_seq }
107   {
108     \pdf_object_ref:n { \_tag/struct/#2 }
109   }
110 }
```

`\cs_generate_variant:Nn _tag_struct_kid_struct_gput_right:nn {xx}`

(End definition for `_tag_struct_kid_struct_gput_right:nn`.)

`_tag_struct_kid_OBJR_gput_right:nn`

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.

`_tag_struct_kid_OBJR_gput_right:xx`

```

113 \cs_new_protected:Npn \_tag_struct_kid_OBJR_gput_right:nn #1 #2 %#1 num of parent struct,
114                                     %#2 obj reference
115 {
116   \pdf_object_unnamed_write:nn
117   { dict }
118   {
119     /Type/OBJR/Obj-#2
120   }
```

```

121     \__tag_seq_gput_right:cx
122     { g__tag_struct_kids_#1_seq }
123     {
124         \pdf_object_ref_last:
125     }
126 }
127
128 \cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nn { xx }
129

```

(End definition for __tag_struct_kid_OBJR_gput_right:nn.)

__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.

```

130 \cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
131 {
132     \seq_gpop_left:NN #1 \l__tag_tmpa_tl
133     \regex_replace_once:nnN
134     { \c{\__tag_mc_insert_mcid_kids:n} }
135     { \c{\__tag_mc_insert_mcid_single_kids:n} }
136     \l__tag_tmpa_tl
137     \seq_gput_left:NV #1 \l__tag_tmpa_tl
138 }
139
140 \cs_generate_variant:Nn\__tag_struct_exchange_kid_command:N { c }

```

(End definition for __tag_struct_exchange_kid_command:N.)

__tag_struct_fill_kid_key:n

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

```

141 \cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
142 {
143     \int_case:nnF
144     {
145         \seq_count:c
146         {
147             g__tag_struct_kids_#1_seq
148         }
149     }
150     {
151         { 0 }
152         { } %no kids, do nothing
153         { 1 } % 1 kid, insert
154         {
155             % in this case we need a special command in
156             % luamode to get the array right. See issue #13
157             \bool_if:NT\g__tag_mode_lua_bool
158             {
159                 \__tag_struct_exchange_kid_command:c
160                 {g__tag_struct_kids_#1_seq}
161             }
162             \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}

```

```

163         {
164             \seq_item:cn
165             {
166                 g__tag_struct_kids_#1_seq
167             }
168             {1}
169         }
170     } %
171 }
172 { %many kids, use an array
173     \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
174     {
175         [
176             \seq_use:cn
177             {
178                 g__tag_struct_kids_#1_seq
179             }
180             {
181                 \c_space_tl
182             }
183         ]
184     }
185 }
186 }
187

```

(End definition for __tag_struct_fill_kid_key:n.)

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

```

188 \cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
189 {
190     \tl_clear:N #2
191     \seq_map_inline:cn
192     {
193         c__tag_struct_
194         \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
195         _entries_seq
196     }
197     {
198         \tl_put_right:Nx
199         #2
200         {
201             \prop_if_in:cnT
202             { g__tag_struct_#1_prop }
203             { ##1 }
204             {
205                 \c_space_tl/##1~\prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
206             }
207         }
208     }
209 }

```

(End definition for __tag_struct_get_dict_content: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.

```

210 \cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
211 {
212   \pdf_object_if_exist:nTF { __tag/struct/#1 }
213   {
214     \__tag_struct_fill_kid_key:n { #1 }
215     \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
216     \exp_args:Nx
217       \pdf_object_write:nx
218       { __tag/struct/#1 }
219       {
220         \l__tag_tmpa_tl
221       }
222   }
223   {
224     \msg_error:nnn { tag } { struct-no-objnum } { #1 }
225   }
226 }

```

(End definition for `__tag_struct_write_obj:n`.)

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

Annotations used as structure content must

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

For a link this looks like this

```

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

```

```

227 \cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 % #1 object reference to the annotat
228                                     % #2 structparent number
229 {
230   \bool_if:NT \g__tag_active_struct_bool
231   {
232     %get the number of the parent structure:
233     \seq_get:NNF
234       \g__tag_struct_stack_seq
235       \l__tag_struct_stack_parent_tmpa_tl
236     {

```

```

237         \msg_error:nn { tag } { struct-faulty-nesting }
238     }
239     %put the obj number of the annot in the kid entry, this also creates
240     %the OBJR object
241     \__tag_struct_kid_OBJR_gput_right:xx
242     {
243         \l__tag_struct_stack_parent_tmpa_tl
244     }
245     {
246         #1 %
247     }
248     % add the parent obj number to the parent tree:
249     \exp_args:Nnx
250     \__tag_parenttree_add_objr:nn
251     {
252         #2
253     }
254     {
255         \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
256     }
257     % increase the int:
258     \stepcounter{ g__tag_parenttree_obj_int }
259 }
260 }

```

(End definition for __tag_struct_insert_annot:nn.)

__tag_get_data_struct_tag: this command allows \tag_get:n to get the current structure tag with the keyword **struct_tag**. We will need to handle nesting

```

261 \cs_new:Npn \__tag_get_data_struct_tag:
262 {
263     \exp_args:Ne
264     \tl_tail:n
265     {
266         \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
267     }
268 }

```

(End definition for __tag_get_data_struct_tag:.)

5 Keys

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

```

label
stash 269 \keys_define:nn { __tag / struct }
tag    270 {
title  271     label .tl_set:N      = \l__tag_struct_key_label_tl,
title-o 272     stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
alttext 273     tag .code:n        = % S property
alttext-o 274     {
actualtext 275         \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Nn\g__tag_role_tags_NS_prop{
actualtext-o_uuuu\lang
ref
E

```



```

276 \tl_gset:Nx \g__tag_struct_tag_tl { \seq_item:Nn\l__tag_tmpa_seq {1} }
277 \tl_gset:Nx \g__tag_struct_tag_NS_tl { \seq_item:Nn\l__tag_tmpa_seq {2} }
278 \__tag_check_structure_tag:N \g__tag_struct_tag_tl
279 \__tag_prop_gput:cnx
280 { \g__tag_struct_int_eval:n {\c@g__tag_struct_abs_int}_prop }
281 { S }
282 { \pdf_name_from_unicode_e:n{ \g__tag_struct_tag_tl} } %
283 \prop_get:NVNT \g__tag_role_NS_prop\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl
284 {
285 \__tag_prop_gput:cnx
286 { \g__tag_struct_int_eval:n {\c@g__tag_struct_abs_int}_prop }
287 { NS }
288 { \l__tag_tmpa_tl } %
289 }
290 },
291 title .code:n = % T property
292 {
293 \str_set_convert:Nnon
294 \l__tag_tmpa_str
295 { #1 }
296 { default }
297 { utf16/hex }
298 \__tag_prop_gput:cnx
299 { \g__tag_struct_int_eval:n {\c@g__tag_struct_abs_int}_prop }
300 { T }
301 { <\l__tag_tmpa_str> }
302 },
303 title-o .code:n = % T property
304 {
305 \str_set_convert:Nnon
306 \l__tag_tmpa_str
307 { #1 }
308 { default }
309 { utf16/hex }
310 \__tag_prop_gput:cnx
311 { \g__tag_struct_int_eval:n {\c@g__tag_struct_abs_int}_prop }
312 { T }
313 { <\l__tag_tmpa_str> }
314 },
315 alttext .code:n = % Alt property
316 {
317 \str_set_convert:Nnon
318 \l__tag_tmpa_str
319 { #1 }
320 { default }
321 { utf16/hex }
322 \__tag_prop_gput:cnx
323 { \g__tag_struct_int_eval:n {\c@g__tag_struct_abs_int}_prop }
324 { Alt }
325 { <\l__tag_tmpa_str> }
326 },
327 alttext-o .code:n = % Alt property
328 {
329 \str_set_convert:Noon

```

```

330         \l__tag_tmpa_str
331         { #1 }
332         { default }
333         { utf16/hex }
334     \__tag_prop_gput:cnx
335     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
336     { Alt }
337     { <\l__tag_tmpa_str> }
338 },
339 actualtext .code:n = % ActualText property
340 {
341     \str_set_convert:Nnon
342     \l__tag_tmpa_str
343     { #1 }
344     { default }
345     { utf16/hex }
346     \__tag_prop_gput:cnx
347     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
348     { ActualText }
349     { <\l__tag_tmpa_str>}
350 },
351 actualtext-o .code:n = % ActualText property
352 {
353     \str_set_convert:Noon
354     \l__tag_tmpa_str
355     { #1 }
356     { default }
357     { utf16/hex }
358     \__tag_prop_gput:cnx
359     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
360     { ActualText }
361     { <\l__tag_tmpa_str>}
362 },
363 lang .code:n = % Lang property
364 {
365     \__tag_prop_gput:cnx
366     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
367     { Lang }
368     { (#1) }
369 },
370 ref .code:n = % Lang property
371 {
372     \tl_clear:N\l__tag_tmpa_tl
373     \clist_map_inline:nn {#1}
374     {
375         \tl_put_right:Nx \l__tag_tmpa_tl
376         {~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} }
377     }
378     \__tag_prop_gput:cnx
379     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
380     { Ref }
381     { [\l__tag_tmpa_tl] }
382 },
383 E .code:n = % E property

```

```

384     {
385       \str_set_convert:Nnon
386       \l__tag_tmpa_str
387       { #1 }
388       { default }
389       { utf16/hex }
390       \__tag_prop_gput:cnx
391       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
392       { E }
393       { <\l__tag_tmpa_str> }
394     },
395   }

```

(End definition for `label` and others. These functions are documented on page 63.)

AF keys for the AF keys (associated files). They use commands from `l3pdffile`! The stream
AFinline variants use `txt` as extension to get the mimetype. TODO: check if this should be
AFinline-o configurable. For math we will perhaps need another extension.

```

396 \keys_define:nn { __tag / struct }
397 {
398   AF .code:n          = % AF property
399   {
400     \pdf_object_if_exist:nTF {#1}
401     {
402       \__tag_prop_gput:cnx
403       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
404       { AF }
405       { \pdf_object_ref:n {#1} }
406     }
407   }
408 }
409 },
410 ,AFinline .code:n =
411 {
412   \group_begin:
413   \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
414   {
415     \pdffile_embed_stream:nxx
416     {#1}
417     {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
418     {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
419   }
420   \__tag_prop_gput:cnx
421   { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
422   { AF }
423   { \pdf_object_ref:e {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int} } }
424   \group_end:
425 }
426 ,AFinline-o .code:n =
427 {
428   \group_begin:
429   \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
430   {
431

```

```

432     \pdffile_embed_stream:oxx
433     {#1}
434     {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
435     {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
436   }
437   \__tag_prop_gput:cnx
438   { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
439   { AF }
440   { \pdf_object_ref:e {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int } }
441   \group_end:
442 }
443 }

```

(End definition for AF, AFinline, and AFinline-o. These functions are documented on page 63.)

6 User commands

```

\tag_struct_begin:n
\tag_struct_end:
444 \cs_new_protected:Npn \tag_struct_begin:n #1 %#1 key-val
445 {
446   \__tag_check_if_active_struct:T
447   {
448     \group_begin:
449     \int_gincr:N \c@g__tag_struct_abs_int
450     \__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
451     \__tag_new_output_prop_handler:n { \int_eval:n { \c@g__tag_struct_abs_int } }
452     \__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq }
453     \exp_args:Ne
454     \pdf_object_new:nn
455     { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
456     { dict }
457     \__tag_prop_gput:cno
458     { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
459     { Type }
460     { /StructElem }
461     \keys_set:nn { __tag / struct } { #1 }
462     \__tag_check_structure_has_tag:n { \int_eval:n { \c@g__tag_struct_abs_int } }
463     \tl_if_empty:NF
464     \l__tag_struct_key_label_tl
465     {
466       \__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_tl}{struct}
467     }
468     %get the potential parent from the stack:
469     \seq_get:NNF
470     \g__tag_struct_stack_seq
471     \l__tag_struct_stack_parent_tmpa_tl
472     {
473       \msg_error:nn { tag } { struct-faulty-nesting }
474     }
475     \seq_gpush:NV \g__tag_struct_stack_seq \c@g__tag_struct_abs_int
476     \seq_gpush:NV \g__tag_struct_tag_stack_seq \g__tag_struct_tag_tl
477     \tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
478     %\seq_show:N \g__tag_struct_stack_seq

```

```

479     \bool_if:NF
480     \l__tag_struct_elem_stash_bool
481     {%set the parent
482     \__tag_prop_gput:cnx
483     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
484     { P }
485     {
486     \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
487     }
488     %record this structure as kid:
489     %\tl_show:N \g__tag_struct_stack_current_tl
490     %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
491     \__tag_struct_kid_struct_gput_right:xx
492     { \l__tag_struct_stack_parent_tmpa_tl }
493     { \g__tag_struct_stack_current_tl }
494     %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
495     %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
496     }
497     %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
498     %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
499     \group_end:
500   }
501 }
502
503
504 \cs_new_protected:Nn \tag_struct_end:
505 { %take the current structure num from the stack:
506   %the objects are written later, lua mode hasn't all needed info yet
507   %\seq_show:N \g__tag_struct_stack_seq
508   \__tag_check_if_active_struct:T
509   {
510     \seq_gpop:NN \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
511     \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
512     {
513       \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
514     }
515     { \__tag_check_no_open_struct: }
516     % get the previous one, shouldn't be empty as the root should be there
517     \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
518     {
519       \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
520     }
521     {
522       \__tag_check_no_open_struct:
523     }
524     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
525     {
526       \tl_gset:NV \g__tag_struct_tag_tl \l__tag_tmpa_tl
527     }
528   }
529 }

```

(End definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 62.)

\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.

```

530 \cs_new_protected:Nn \tag_struct_use:n {%#1 is the label
531 {
532   \__tag_check_if_active_struct:T
533   {
534     \prop_if_exist:cTF
535     { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
536     {
537       \__tag_check_struct_used:n {#1}
538       %add the label structure as kid to the current structure (can be the root)
539       \__tag_struct_kid_struct_gput_right:xx
540       { \g__tag_struct_stack_current_tl }
541       { \__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0} }
542       %add the current structure to the labeled one as parents
543       \__tag_prop_gput:cnx
544       { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
545       { P }
546       {
547         \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
548       }
549     }
550   }
551   \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
552 }
553 }
554 }

```

(End definition for \tag_struct_use:n. This function is documented on page 62.)

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

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

```

555 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 {%#1 should be an object reference
556                                     %#2 struct parent num
557 {
558   \__tag_check_if_active_struct:T
559   {
560     \__tag_struct_insert_annot:nn {#1}{#2}
561   }
562 }
563
564 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx}
565 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
566
567 \</package>
568

```

(End definition for \tag_struct_insert_annot:nn and \tag_struct_parent_int:. These functions are documented on page 62.)

7 Attributes and attribute classes

```

569 \<header>
570 \ProvidesExplPackage {tagpdf-attr-code} {2021-07-03} {0.91}
571 {part of tagpdf - code related to attributes and attribute classes}
572 \</header>

```

7.1 Variables

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

```

573 \<package>
574 \prop_new:N \g__tag_attr_entries_prop
575 \seq_new:N \g__tag_attr_class_used_seq
576 \tl_new:N \l__tag_attr_value_tl
577 \prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes

(End definition for \g__tag_attr_entries_prop and others.)

```

7.2 Commands and keys

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

```

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

578 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
579 {
580   \prop_gput:Nnn \g__tag_attr_entries_prop
581     {#1}{#2}
582 }
583
584 \keys_define:nn { __tag / setup }
585 {
586   newattribute .code:n =
587     {
588       \__tag_attr_new_entry:nn #1
589     }
590 }

```

(End definition for `__tag_attr_new_entry:nn` and `newattribute`. This function is documented on page 64.)

attribute-class attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

```

591 \keys_define:nn { __tag / struct }
592 {
593   attribute-class .code:n =
594   {
595     \clist_set:No \l__tag_tmpa_clist { #1 }
596     \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
597     \seq_map_inline:Nn \l__tag_tmpa_seq
598     {
599       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
600       {
601         \msg_error:nnn { tag } { attr-unknown } { ##1 }
602       }
603       \seq_gput_left:Nn\g__tag_attr_class_used_seq { ##1}
604     }
605     \seq_set_map:NNn \l__tag_tmpb_seq \l__tag_tmpa_seq
606     {
607       /##1
608     }
609     \tl_set:Nx \l__tag_tmpa_tl
610     {
611       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
612       \seq_use:Nn \l__tag_tmpb_seq { \c_space_tl }
613       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
614     }
615     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
616     {
617       __tag_prop_gput:cnx
618       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
619       { C }
620       { \l__tag_tmpa_tl }
621       %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
622     }
623   }
624 }

```

(End definition for attribute-class. This function is documented on page 64.)

attribute

```

625 \keys_define:nn { __tag / struct }
626 {
627   attribute .code:n = % A property (attribute, value currently a dictionary)
628   {
629     \clist_set:No \l__tag_tmpa_clist { #1 }
630     \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
631     \tl_set:Nx \l__tag_attr_value_tl
632     {
633       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
634     }
635     \seq_map_inline:Nn \l__tag_tmpa_seq
636     {
637       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
638       {

```



```

639         \msg_error:nnn { tag } { attr-unknown } { ##1 }
640     }
641     \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
642     {%\prop_show:N \g__tag_attr_entries_prop
643       \pdf_object_unnamed_write:nx
644       { dict }
645       {
646         \prop_item:Nn\g__tag_attr_entries_prop {##1}
647       }
648       \prop_gput:Nnx \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
649     }
650     \tl_put_right:Nx \l__tag_attr_value_tl
651     {
652       \c_space_tl
653       \prop_item:Nn \g__tag_attr_objref_prop {##1}
654     }
655     % \tl_show:N \l__tag_attr_value_tl
656     }
657     \tl_put_right:Nx \l__tag_attr_value_tl
658     { %[
659       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
660     }
661     % \tl_show:N \l__tag_attr_value_tl
662     \__tag_prop_gput:cnx
663     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
664     { A }
665     { \l__tag_attr_value_tl }
666   },
667 }
668 \end{package}

```

(End definition for attribute. This function is documented on page 64.)

Part VIII

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2021-07-03} {0.91}
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
16 \cs_set_protected:Npn \__tag_seq_new:N #1
17 {
18   \seq_new:N #1
19   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
20 }
21
22
23 \cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
24 {
25   \prop_gput:Nnn #1 { #2 } { #3 }
26   \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 ["#2"] = "#3" }
27 }
28
29
```

```

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

```

(End definition for __tag_prop_new:N and others.)

62 \langle /luatex \rangle

The module declaration

```

63  $\langle$ *lua $\rangle$ 
64 -- tagpdf.lua
65 -- Ulrike Fischer
66
67 local ProvidesLuaModule = {
68   name      = "tagpdf",
69   version   = "0.91",      --TAGVERSION
70   date      = "2021-07-03", --TAGDATE
71   description = "tagpdf lua code",
72   license    = "The LATEX Project Public License 1.3c"
73 }
74
75 if luatexbase and luatexbase.provides_module then
76   luatexbase.provides_module (ProvidesLuaModule)
77 end
78
79 --[[

```

```

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

```

Some comments about the lua structure.

```

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

```

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

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

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

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

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

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

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

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

```
161 ltx          = ltx          or { }
162 ltx.__tag     = ltx.__tag     or { }
163 ltx.__tag.mc   = ltx.__tag.mc   or { } -- mc data
164 ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
165 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
166                                     -- wasn't a so great idea ...
167                                     -- g__tag_role_tags_seq used by tag<-> is in this tabl
168 ltx.__tag.page  = ltx.__tag.page  or { } -- page data, currently only i->{0->mcnum,1->mcr
169 ltx.__tag.trace = ltx.__tag.trace or { } -- show commands
170 ltx.__tag.func  = ltx.__tag.func  or { } -- functions
```

```
171 ltx.__tag.conf      = ltx.__tag.conf  or { } -- configuration variables
```

2 Logging functions

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

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

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

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

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

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

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

3 Helper functions

3.1 Retrieve data functions

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

```

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

```

(End definition for `__tag_get_mc_cnt_type_tag`.)

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

```

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

```

(End definition for `__tag_get_mathsubtype`.)

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

```

258 local __tag_get_num_from =
259   function (tag)
260     if ltx.__tag.tables["g__tag_role_tags_prop"][tag] then
261       a= ltx.__tag.tables["g__tag_role_tags_prop"][tag]
262     else
263       a= -1
264     end
265     return a
266   end
267
268 ltx.__tag.func.get_num_from = __tag_get_num_from
269
270 function ltx.__tag.func.output_num_from (tag)
271   local num = __tag_get_num_from (tag)
272   tex.sprint(catlatex,num)
273   if num == -1 then
274     __tag_log ("Unknown tag "..tag.." used")
275   end
276 end

```

(End definition for `__tag_get_num_from`, `ltx.__tag.func.get_num_from`, and `ltx.__tag.func.output_num_from`.)

<pre> __tag_get_tag_from ltx.__tag.func.get_tag_from ltx.__tag.func.output_tag_from </pre>	<p>These functions are the opposites to the previous function: they take as argument a number (the attribute value) and return the string <code>tag</code>. The first function outputs the number for lua, while the <code>output</code> function outputs to tex.</p>
	<pre> 277 local __tag_get_tag_from = 278 function (num) 279 if ltx.__tag.tables["g__tag_role_tags_seq"][num] then 280 a = ltx.__tag.tables["g__tag_role_tags_seq"][num] 281 else 282 a = "UNKNOWN" 283 end 284 return a 285 end 286 287 ltx.__tag.func.get_tag_from = __tag_get_tag_from 288 289 function ltx.__tag.func.output_tag_from (num) 290 tex.sprint(catlatex,__tag_get_tag_from (num)) 291 end </pre> <p>(End definition for <code>__tag_get_tag_from</code>, <code>ltx.__tag.func.get_tag_from</code>, and <code>ltx.__tag.func.output_tag_from</code>.)</p>
<pre>ltx.__tag.func.store_mc_data</pre>	<p>This function stores for <code>key=data</code> for mc-chunk <code>num</code>. It is used in the <code>tagpdf-mc</code> code, to store for example the tag string, and the raw options.</p> <pre> 292 function ltx.__tag.func.store_mc_data (num,key,data) 293 ltx.__tag.mc[num] = ltx.__tag.mc[num] or { } 294 ltx.__tag.mc[num][key] = data 295 __tag_log ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).. " => "..tostring(data),3) 296 end </pre> <p>(End definition for <code>ltx.__tag.func.store_mc_data</code>.)</p>
<pre>ltx.__tag.func.store_mc_label</pre>	<p>This function stores the <code>label=num</code> relationship in the <code>labels</code> subtable. TODO: this is probably unused and can go.</p> <pre> 297 function ltx.__tag.func.store_mc_label (label,num) 298 ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { } 299 ltx.__tag.mc.labels[label] = num 300 end </pre> <p>(End definition for <code>ltx.__tag.func.store_mc_label</code>.)</p>
<pre>ltx.__tag.func.store_mc_kid</pre>	<p>This function is used in the traversing code. It stores a sub-chunk of a mc <code>mcnum</code> into the <code>kids</code> table.</p> <pre> 301 function ltx.__tag.func.store_mc_kid (mcnum,kid,page) 302 ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3) 303 ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { } 304 local kidtable = {kid=kid,page=page} 305 tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable) 306 end </pre> <p>(End definition for <code>ltx.__tag.func.store_mc_kid</code>.)</p>

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

```

307 function ltx.__tag.func.mc_num_of_kids (mcnum)
308   local num = 0
309   if ltx.__tag.mc[mcnun] and ltx.__tag.mc[mcnun]["kids"] then
310     num = #ltx.__tag.mc[mcnun]["kids"]
311   end
312   ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnun .. "has " .. num .. "KIDS",4)
313   return num
314 end

```

(End definition for ltx.__tag.func.mc_num_of_kids.)

3.2 Functions to insert the pdf literals

__tag_insert_emc_node This insert the emc node.

```

315 local function __tag_insert_emc_node (head,current)
316   local emcnode = nodenew("whatsit","pdf_literal")
317     emcnode.data = "EMC"
318     emcnode.mode=1
319     head = node.insert_before(head,current,emcnode)
320   return head
321 end

```

(End definition for __tag_insert_emc_node.)

__tag_insert_bmc_node This inserts a simple bmc node

```

322 local function __tag_insert_bmc_node (head,current,tag)
323   local bmcnode = nodenew("whatsit","pdf_literal")
324     bmcnode.data = "/"..tag.." BMC"
325     bmcnode.mode=1
326     head = node.insert_before(head,current,bmcnode)
327   return head
328 end

```

(End definition for __tag_insert_bmc_node.)

__tag_insert_bdc_node This inserts a bcd node with a fix dict. TODO: check if this is still used, now that we create properties.

```

329 local function __tag_insert_bdc_node (head,current,tag,dict)
330   local bdcnode = nodenew("whatsit","pdf_literal")
331     bdcnode.data = "/"..tag.."<<..dict..>> BDC"
332     bdcnode.mode=1
333     head = node.insert_before(head,current,bdcnode)
334   return head
335 end

```

(End definition for __tag_insert_bdc_node.)

__tag_pdf_object_ref This allows to reference a pdf object reserved with the l3pdf command by name. The return value is n 0 R, if the object doesn't exist, n is 0. TODO: is uses internal l3pdf commands, this should be properly supported by l3pdf

```

336 local function __tag_pdf_object_ref (name)
337   local tokename = 'c_pdf_backend_object_'..name..'_int'

```

```

338     local object = token.create(tokenname).index..' 0 R'
339     return object
340 end
341 ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref

(End definition for __tag_pdf_object_ref and ltx.__tag.func.pdf_object_ref.)

```

4 Function for the real space chars

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

```

342 local function __tag_show_spacemark (head,current,color,height)
343     local markcolor = color or "1 0 0"
344     local markheight = height or 10
345     local pdfstring = node.new("whatsit","pdf_literal")
346         pdfstring.data =
347             string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
348                 3,markheight)
349             head = node.insert_after(head,current,pdfstring)
350     return head
351 end

(End definition for __tag_show_spacemark.)

```

`__tag_fakespace` This is used to define a lua version of `\pdf-fakespace`

```

ltx.__tag.func.fakespace 351 local function __tag_fakespace()
352     tex.setattribute(iwspaceattributeid,1)
353     tex.setattribute(iwfontattributeid,font.current())
354 end
355 ltx.__tag.func.fakespace = __tag_fakespace

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

```

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

```

356 --[[ a function to mark up places where real space chars should be inserted
357     it only sets an attribute.
358 --]]
359
360 local function __tag_mark_spaces (head)
361     local inside_math = false
362     for n in nodetraverse(head) do
363         local id = n.id
364         if id == GLYPH then
365             local glyph = n
366             if glyph.next and (glyph.next.id == GLUE)
367                 and not inside_math and (glyph.next.width > 0)
368             then
369                 nodesetattribute(glyph.next,iwspaceattributeid,1)
370                 nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
371             -- for debugging
372             if ltx.__tag.trace.showspace then

```

```

373     __tag_show_spacemark (head, glyph)
374   end
375   elseif glyph.next and (glyph.next.id == KERN) and not inside_math then
376     local kern = glyph.next
377     if kern.next and (kern.next.id == GLUE) and (kern.next.width > 0)
378     then
379       nodesetattribute(kern.next, iwspaceattributeid, 1)
380       nodesetattribute(kern.next, iwfontattributeid, glyph.font)
381     end
382   end
383   -- look also back
384   if glyph.prev and (glyph.prev.id == GLUE)
385     and not inside_math
386     and (glyph.prev.width > 0)
387     and not nodehasattribute(glyph.prev, iwspaceattributeid)
388   then
389     nodesetattribute(glyph.prev, iwspaceattributeid, 1)
390     nodesetattribute(glyph.prev, iwfontattributeid, glyph.font)
391   -- for debugging
392     if ltx.__tag.trace.showspace then
393       __tag_show_spacemark (head, glyph)
394     end
395   end
396   elseif id == PENALTY then
397     local glyph = n
398     -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty, 3)
399     if glyph.next and (glyph.next.id == GLUE)
400       and not inside_math and (glyph.next.width > 0) and n.subtype == 0
401     then
402       nodesetattribute(glyph.next, iwspaceattributeid, 1)
403       -- nodesetattribute(glyph.next, iwfontattributeid, glyph.font)
404     -- for debugging
405       if ltx.__tag.trace.showspace then
406         __tag_show_spacemark (head, glyph)
407       end
408     end
409     elseif id == MATH then
410       inside_math = (n.subtype == 0)
411     end
412   end
413   return head
414 end

```

(End definition for __tag_mark_spaces.)

```

__tag_activate_mark_space  These functions add/remove the function which marks the spaces to the callbacks
ltx.__tag.func.markspaceon pre_linebreak_filter and hpack_filter
ltx.__tag.func.markspaceoff
415 local function __tag_activate_mark_space ()
416   if not luatexbase.in_callback ("pre_linebreak_filter", "markspaces") then
417     luatexbase.add_to_callback("pre_linebreak_filter", __tag_mark_spaces, "markspaces")
418     luatexbase.add_to_callback("hpack_filter", __tag_mark_spaces, "markspaces")
419   end
420 end
421

```

```

422 ltx.__tag.func.markspaceon=__tag_activate_mark_space
423
424 local function __tag_deactivate_mark_space ()
425 if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
426 luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
427 luatexbase.remove_from_callback("hpack_filter","markspaces")
428 end
429 end
430
431 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space

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

```

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

```

default_fontid 432 local default_space_char = node.new(GLYPH)
433 local default_fontid = font.id("TU/lmr/m/n/10")
434 default_space_char.char = 32
435 default_space_char.font = default_fontid

```

(End definition for default_space_char and default_fontid. These functions are documented on page ??.)

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

```

436 local function __tag_space_chars_shipout (box)
437 local head = box.head
438 if head then
439 for n in node.traverse(head) do
440 local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
441 if n.id == HLIST then -- enter the hlist
442 __tag_space_chars_shipout (n)
443 elseif n.id == VLIST then -- enter the vlist
444 __tag_space_chars_shipout (n)
445 elseif n.id == GLUE then
446 if ltx.__tag.trace.showspaces and spaceattr==1 then
447 __tag_show_spacemark (head,n,"0 1 0")
448 end
449 if spaceattr==1 then
450 local space
451 local space_char = node.copy(default_space_char)
452 local curfont = nodegetattribute(n,iwfontattributeid)
453 ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
454 if curfont and luaotfload.aux.slot_of_name(curfont,"space") then
455 space_char.font=curfont
456 end
457 head, space = node.insert_before(head, n, space_char) --
458 n.width = n.width - space.width
459 space.attr = n.attr
460 end
461 end
462 end
463 end
464 end

```

```

465
466 function ltx.__tag.func.space_chars_shipout (box)
467   __tag_space_chars_shipout (box)
468 end

```

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

5 Function for the tagging

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

```

469 function ltx.__tag.func.mc_insert_kids (mcnum,single)
470   if ltx.__tag.mc[mcnum] then
471     ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
472     if ltx.__tag.mc[mcnum]["kids"] then
473       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
474         tex.sprint("[")
475       end
476       for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
477         local kidnum = kidstable["kid"]
478         local kidpage = kidstable["page"]
479         local kidpageobjnum = pdfpageref(kidpage)
480         ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
481           " insert KID " .. i..
482           " with num " .. kidnum ..
483           " on page " .. kidpage.."/"..kidpageobjnum,3)
484         tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> " )
485       end
486       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
487         tex.sprint("]")
488       end
489     else
490       -- this is typically not a problem, e.g. empty hbox in footer/header can
491       -- trigger this warning.
492       ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
493       if single==1 then
494         tex.sprint("null")
495       end
496     end
497   else
498     ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
499   end
500 end

```

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

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

```

501 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
502   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
503   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }

```

```

504 -- a structure can contain more than on mc chunk, the content should be ordered
505 tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
506 ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
507     mcnum.." inserted in struct "..structnum,3)
508 -- but every mc can only be in one structure
509 ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
510 ltx.__tag.mc[mcnum]["parent"] = structnum
511 end
512

```

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

ltx.__tag.func.store_mc_in_page This is used in the traversing code and stores the relation between abs count and page count.

```

513 -- pay attention: lua counts arrays from 1, tex pages from one
514 -- mcid and arrays in pdf count from 0.
515 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagencnt,page)
516     ltx.__tag.page[page] = ltx.__tag.page[page] or {}
517     ltx.__tag.page[page][mcpagencnt] = mcnum
518     ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
519         ": inserting MCID " .. mcpagencnt .. " => " .. mcnum,3)
520 end

```

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

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

```

521 --[[
522     Now follows the core function
523     It wades through the shipout box and checks the attributes
524     ARGUMENTS
525     box: is a box,
526     mcpagencnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
527     mcntprev: num, the attribute cnt of the previous node/whatever - if different we have a c
528     mcopy: num, records if some bdc/emc is open
529     These arguments are only needed for log messages, if not present are replaces by fix string
530     name: string to describe the box
531     mctypeprev: num, the type attribute of the previous node/whatever
532
533     there are lots of logging messages currently. Should be cleaned up in due course.
534     One should also find ways to make the function shorter.
535 --]]
536
537 function ltx.__tag.func.mark_page_elements (box,mcpagencnt,mcntprev,mcopy,name,mctypeprev)
538     local name = name or ("SOMEBOX")
539     local mctypeprev = mctypeprev or -1
540     local abspage = status.total_pages + 1 -- the real counter is increased
541                                           -- inside the box so one off
542                                           -- if the callback is not used. (???)
543     ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
544     ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagencnt..
545         " prev "..mcntprev ..
546         " type prev "..mctypeprev,4)
547     ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
548         " TYPE ".. node.type(node.getid(box)),3)

```

```

549 local head = box.head -- ShipoutBox is a vlist?
550 if head then
551     mccnthead, mctypehead, taghead = __tag_get_mc_cnt_type_tag (head)
552     ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
553                         node.type(node.getid(head))..
554                         " MC"..tostring(mccnthead)..
555                         " => TAG " .. tostring(mctypehead)..
556                         " => ".. tostring(taghead),3)
557 else
558     ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
559                         tostring(head),3)
560 end
561 for n in node.traverse(head) do
562     local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
563     local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
564     ltx.__tag.trace.log ("INFO TAG-NODE: "..
565                         node.type(node.getid(n))..
566                         " MC".. tostring(mccnt)..
567                         " => TAG ".. tostring(mctype)..
568                         " => " .. tostring(tag),3)
569     if n.id == HLIST
570     then -- enter the hlist
571         mcopen,mcpagecnt,mccntprev,mctypeprev=
572         ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypeprev)
573     elseif n.id == VLIST then -- enter the vlist
574         mcopen,mcpagecnt,mccntprev,mctypeprev=
575         ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypeprev)
576     elseif n.id == GLUE then -- at glue real space chars are inserted, but this has
577                             -- been done if the previous shipout wandering, so here it
578     elseif n.id == LOCAL_PAR then -- local_par is ignored
579     elseif n.id == PENALTY then -- penalty is ignored
580     elseif n.id == KERN then -- kern is ignored
581         ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
582                             node.type(node.getid(n)).." " ..n.subtype,4)
583     else
584         -- math is currently only logged.
585         -- we could mark the whole as math
586         -- for inner processing the mlist_to_hlist callback is probably needed.
587         if n.id == MATH then
588             ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
589                             node.type(node.getid(n)).." " ..__tag_get_mathsubtype(n),4)
590         end
591         -- endmath
592         ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
593                             mccnt.." prev "..mccntprev,4)
594         if mccnt~=mccntprev then -- a new mc chunk
595             ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
596                                 node.type(node.getid(n))..
597                                 " MC"..tostring(mccnt)..
598                                 " <=> PREVIOUS "..tostring(mccntprev),4)
599         if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
600             box.list=__tag_insert_emc_node (box.list,n)
601             mcopen = mcopen - 1
602             ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..

```



```

603         mcpagecnt .. " MCOPEN = " .. mcpopen,3)
604     if mcpopen ~=0 then
605         ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcpopen,1)
606     end
607 end
608 if ltx.__tag.mc[mccnt] then
609     if ltx.__tag.mc[mccnt]["artifact"] then
610         ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
611             tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
612         if ltx.__tag.mc[mccnt]["artifact"] == "" then
613             box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
614         else
615             box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]
616         end
617     else
618         ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
619             tostring(tag),3)
620         mcpagecnt = mcpagecnt +1
621         ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
622         local dict= "/MCID "..mcpagecnt
623         if ltx.__tag.mc[mccnt]["raw"] then
624             ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
625                 tostring(ltx.__tag.mc[mccnt]["raw"]),3)
626             dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
627         end
628         if ltx.__tag.mc[mccnt]["alt"] then
629             ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
630                 tostring(ltx.__tag.mc[mccnt]["alt"]),3)
631             dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
632         end
633         if ltx.__tag.mc[mccnt]["actualtext"] then
634             ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
635                 tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
636             dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
637         end
638         box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
639         ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
640         ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
641         ltx.__tag.trace.show_mc_data (mccnt,3)
642     end
643     mcpopen = mcpopen + 1
644 else
645     if tagunmarkedbool.mode == truebool.mode then
646         ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
647         box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
648         mcpopen = mcpopen + 1
649     else
650         ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
651     end
652 end
653 mccntprev = mccnt
654 end
655 end -- end if
656 end -- end for

```

```

657 if head then
658     mccnthead, mctypehead, taghead = __tag_get_mc_cnt_type_tag (head)
659     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
660         node.type(node.getid(head))..
661         " MC"..tostring(mccnthead)..
662         " => TAG "..tostring(mctypehead)..
663         " => "..tostring(taghead),4)
664 else
665     ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
666 end
667 ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
668     tostring(name)..
669     " TYPE ".. node.type(node.getid(box)),4)
670 return mcopen, mcpagecnt, mccntprev, mctypeprev
671 end
672

```

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

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

```

673 function ltx.__tag.func.mark_shipout (box)
674     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
675     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
676         local emcnode = nodenew("whatsit","pdf_literal")
677         local list = box.list
678         emcnode.data = "EMC"
679         emcnode.mode=1
680         if list then
681             list = node.insert_after (list,node.tail(list),emcnode)
682             mcopen = mcopen - 1
683             ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
684         else
685             ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
686         end
687         if mcopen ~=0 then
688             ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
689         end
690     end
691 end

```

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

6 Parenttree

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

ltx.__tag.func.output_parenttree

```

692 function ltx.__tag.func.fill_parent_tree_line (page)
693     -- we need to get page-> i=kid -> mcnum -> structnum
694     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
695     local numsentry = ""
696     local pdfpage = page-1

```

```

697 if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
698   mcchunks=#ltx.__tag.page[page]
699   ltx.__tag.trace.log("INFO PARENTTREE-NUM: page "..
700     page.." has "..mcchunks.." +1 Elements ",4)
701   for i=0,mcchunks do
702     -- what does this log??
703     ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: "..
704       ltx.__tag.page[page][i],4)
705   end
706   if mcchunks == 0 then
707     -- only one chunk so no need for an array
708     local mcnum = ltx.__tag.page[page][0]
709     local structnum = ltx.__tag.mc[mcnum]["parent"]
710     local propname = "g__tag_struct"..structnum.."_prop"
711     --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
712     local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
713     ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF: =====>.."
714       tostring(objref),5)
715     numsentry = pdfpage .. " [".. objref .. "]"
716     ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
717       page.. " num entry = ".. numsentry,3)
718   else
719     numsentry = pdfpage .. " ["
720     for i=0,mcchunks do
721       local mcnum = ltx.__tag.page[page][i]
722       local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
723       local propname = "g__tag_struct"..structnum.."_prop"
724       --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
725       local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
726       numsentry = numsentry .. " " .. objref
727     end
728     numsentry = numsentry .. "]"
729     ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
730       page.. " num entry = ".. numsentry,3)
731   end
732   else
733     ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
734   end
735   return numsentry
736 end
737
738 function ltx.__tag.func.output_parenttree (abspage)
739   for i=1,abspage do
740     line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
741     tex.sprint(catlatex,line)
742   end
743 end

```

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

```

744 </lua>

```

Part IX

The tagpdf-roles module

Tags, roles and namespace code

Part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2021-07-03} {0.91}
4 {part of tagpdf - code related to roles and structure names}
5 </header>
```

1 Code related to roles and structure names

1.1 Variables

Tags have both a name (a string) and a number (for the lua attribute). Testing a name is easier with a prop, while accessing with a number is better done with a seq. So both are used and must be kept in sync if a new tag is added. The number is only relevant for the MC type, tags with the same name from different names spaces can have the same number.

```
\g__tag_role_tags_seq
\g__tag_role_tags_prop 6 <*package>
7 \__tag_seq_new:N \g__tag_role_tags_seq %to get names (type/NS) from numbers
8 \__tag_prop_new:N \g__tag_role_tags_prop %to get numbers from names (type/NS)
(End definition for \g__tag_role_tags_seq and \g__tag_role_tags_prop.)
```

\g__tag_role_tags_NS_prop in pdf 2.0 tags belong to a name space. For every tag we store a default name space. The keys are the tags, the value shorthands like pdf2, or mathml. There is no need to access this from lua, so we use the standard prop commands.

```
9 \prop_new:N \g__tag_role_tags_NS_prop %to namespace info
(End definition for \g__tag_role_tags_NS_prop.)
```

\g__tag_role_NS_prop The standard names spaces are the following. The keys are the name tagpdf will use, the urls are the identifier in the namespace object.

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

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

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

user \c__tag_role_userNS_id_str (random id, for user tags)

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

```
10 \prop_new:N \g__tag_role_NS_prop % collect namespaces
```

(End definition for \g__tag_role_NS_prop.)

We need also a bunch of temporary variables:

```
\l__tag_role_tag_tmpa_tl
\l__tag_role_tag_namespace_tmpa_tl
\l__tag_role_role_tmpa_tl
\l__tag_role_role_namespace_tmpa_tl
11 \tl_new:N \l__tag_role_tag_tmpa_tl
12 \tl_new:N \l__tag_role_tag_namespace_tmpa_tl
13 \tl_new:N \l__tag_role_role_tmpa_tl
14 \tl_new:N \l__tag_role_role_namespace_tmpa_tl
```

(End definition for \l__tag_role_tag_tmpa_tl and others.)

1.2 Namespaces

The following commands setups a names space. Namespace dictionaries can contain an optional /Schema and /RoleMapNS entry. We only reserve the objects but delay the writing to the finish code, where we can test if the keys and the name spaces are actually needed This commands setups objects for the name space and its rolemap. It also initialize a prop to collect the rolemaps if needed.

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

```
\__tag_role_NS_new:nnn
15 \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
16 {
17   \pdf_object_new:nn {tag/NS/#1}{dict}
18   \pdfdict_new:n {g__tag_role/Namespace_#1_dict}
19   \pdf_object_new:nn {\__tag_role/RoleMapNS/#1}{dict}
20   \pdfdict_new:n {g__tag_role/RoleMapNS_#1_dict}
21   \pdfdict_gput:nnn
22     {g__tag_role/Namespace_#1_dict}
23     {Type}
24     {/Namespace}
25   \pdf_string_from_unicode:nnN{utf8/string}{#2}\l_tmpa_str
26   \tl_if_empty:NF \l_tmpa_str
27   {
28     \pdfdict_gput:nnx
29       {g__tag_role/Namespace_#1_dict}
30       {NS}
31       {\l_tmpa_str}
32   }
33   %RoleMapNS is added in tree
34   \tl_if_empty:NF {#3}
35   {
36     \pdfdict_gput:nnx{g__tag_role/Namespace_#1_dict}
37       {Schema}{#3}
38   }
39   \prop_gput:Nnx \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
40 }
```

(End definition for __tag_role_NS_new:nnn.)

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

```
\c__tag_role_userNS_id_str
```

```

41 \str_const:Nx \c__tag_role_userNS_id_str
42 { data:,
43   \int_to_Hex:n{\int_rand:n {65535}}
44   \int_to_Hex:n{\int_rand:n {65535}}
45   -
46   \int_to_Hex:n{\int_rand:n {65535}}
47   -
48   \int_to_Hex:n{\int_rand:n {65535}}
49   -
50   \int_to_Hex:n{\int_rand:n {65535}}
51   -
52   \int_to_Hex:n{\int_rand:n {16777215}}
53   \int_to_Hex:n{\int_rand:n {16777215}}
54 }

```

(End definition for \c__tag_role_userNS_id_str.)

Now we setup the standard names spaces. Currently only if we detect pdf2.0 but this will perhaps have to change if the structure code gets to messy.

```

55 \pdf_version_compare:NnT > {1.9}
56 {
57   \__tag_role_NS_new:nnn {pdf}   {http://iso.org/pdf/ssn}{ }
58   \__tag_role_NS_new:nnn {pdf2}  {http://iso.org/pdf2/ssn}{ }
59   \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{ }
60   \exp_args:Nnx
61   \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{ }
62 }

```

1.3 Data

In this section we setup the standard data. At first the list of structure types. We split them in three lists, the tags with which are both in the pdf and pdf2 namespace, the one only in pdf and the one with the tags only in pdf2. We also define a rolemap for the pdfII only type to pdf so that they can always be used.

```

\c__tag_role_sttags_pdf_pdfII_clist
\c__tag_role_sttags_only_pdf_clist
\c__tag_role_sttags_only_pdfII_clist
\c__tag_role_sttags_mathml_clist
\c__tag_role_sttags_pdfII_to_pdf_prop
63 %
64 \clist_const:Nn \c__tag_role_sttags_pdf_pdfII_clist
65 {
66   Document,    %A complete document. This is the root element
67                %of any structure tree containing
68                %multiple parts or multiple articles.
69   Part,        %A large-scale division of a document.
70   Sect,        %A container for grouping related content elements.
71   Div,         %A generic block-level element or group of elements
72   Caption,     %A brief portion of text describing a table or figure.
73   Index,
74   NonStruct,   %probably not needed
75   H,
76   H1,
77   H2,
78   H3,
79   H4,

```

```

80     H5,
81     H6,
82     P,
83     L,          %list
84     LI,         %list item (around label and list item body)
85     Lbl,        %list label
86     LBody,      %list item body
87     Table,
88     TR,         %table row
89     TH,         %table header cell
90     TD,         %table data cell
91     THead,      %table header (n rows)
92     TBody,      %table rows
93     TFoot,      %table footer
94     Span,       %generic inline marker
95     Link,       %
96     Annot,
97     Figure,
98     Formula,
99     Form,
100    % ruby warichu etc ..
101    Ruby,
102    RB,
103    RT,
104    Warichu,
105    WT,
106    WP,
107    Artifact % only MC-tag ?...
108 }
109
110 \clist_const:Nn \c__tag_role_sttags_only_pdf_clist
111 {
112     Art,         %A relatively self-contained body of text
113                 %constituting a single narrative or exposition
114     BlockQuote, %A portion of text consisting of one or more paragraphs
115                 %attributed to someone other than the author of the
116                 %surrounding text.
117     TOC,         %A list made up of table of contents item entries
118                 % (structure tag TOCI; see below) and/or other
119                 % nested table of contents entries
120     TOCI,        %An individual member of a table of contents.
121                 %This entry's children can be any of the following structure tags:
122                 %Lbl,Reference,NonStruct,P,TOC
123     Index,
124     Private,
125     Quote,       %inline quote
126     Note,        %footnote, endnote. Lbl can be child
127     Reference,   %A citation to content elsewhere in the document.
128     BibEntry,    %bibentry
129     Code
130 }
131
132 \clist_const:Nn \c__tag_role_sttags_only_pdfIII_clist
133 {

```

```

134     DocumentFragment
135     ,Aside
136     ,H7
137     ,H8
138     ,H9
139     ,H10
140     ,Title
141     ,FENote
142     ,Sub
143     ,Em
144     ,Strong
145     ,Artifact
146 }
147
148 \clist_const:Nn \c__tag_role_sttags_mathml_clist
149 {
150     abs
151     ,and
152     ,annotation
153     ,apply
154     ,approx
155     ,arccos
156     ,arccosh
157     ,arccot
158     ,arccoth
159     ,arccsc
160     ,arccsch
161     ,arcsec
162     ,arcsech
163     ,arcsin
164     ,arcsinh
165     ,arctan
166     ,arctanh
167     ,arg
168     ,bind
169     ,bvar
170     ,card
171     ,cartesianproduct
172     ,cbytes
173     ,ceiling
174     ,cerror
175     ,ci
176     ,cn
177     ,codomain
178     ,complexes
179     ,compose
180     ,condition
181     ,conjugate
182     ,cos
183     ,cosh
184     ,cot
185     ,coth
186     ,cs
187     ,csc

```



```

188 ,csch
189 ,csymbol
190 ,curl
191 ,declare
192 ,degree
193 ,determinant
194 ,diff
195 ,divergence
196 ,divide
197 ,domain
198 ,domainofapplication
199 ,emptyset
200 ,eq
201 ,equivalent
202 ,eulergamma
203 ,exists
204 ,exp
205 ,exponentiale
206 ,factorial
207 ,factorof
208 ,false
209 ,floor
210 ,fn
211 ,forall
212 ,gcd
213 ,geq
214 ,grad
215 ,gt
216 ,ident
217 ,image
218 ,imaginary
219 ,imaginaryi
220 ,implies
221 ,in
222 ,infinity
223 ,int
224 ,integers
225 ,intersect
226 ,interval
227 ,inverse
228 ,lambda
229 ,laplacian
230 ,lcm
231 ,leq
232 ,limit
233 ,ln
234 ,log
235 ,logbase
236 ,lowlimit
237 ,lt
238 ,maction
239 ,maligngroup
240 ,malignmark
241 ,math

```

242 ,matrix
 243 ,matrixrow
 244 ,max
 245 ,mean
 246 ,median
 247 ,menclase
 248 ,merror
 249 ,mfenced
 250 ,mfrac
 251 ,mglyph
 252 ,mi
 253 ,min
 254 ,minus
 255 ,mlabeledtr
 256 ,mlongdiv
 257 ,mmultiscripts
 258 ,mn
 259 ,mo
 260 ,mode
 261 ,moment
 262 ,momentabout
 263 ,mover
 264 ,mpadded
 265 ,mphantom
 266 ,mprescripts
 267 ,mroot
 268 ,mrow
 269 ,ms
 270 ,mscarries
 271 ,mscarry
 272 ,msgroup
 273 ,msline
 274 ,mspace
 275 ,msqrt
 276 ,msrow
 277 ,mstack
 278 ,mstyle
 279 ,msub
 280 ,msubsup
 281 ,msup
 282 ,mtable
 283 ,mtd
 284 ,mtext
 285 ,mtr
 286 ,munder
 287 ,munderover
 288 ,naturalnumbers
 289 ,neq
 290 ,none
 291 ,not
 292 ,notanumber
 293 ,notin
 294 ,notprsubset
 295 ,notsubset

```

296     ,or
297     ,otherwise
298     ,outerproduct
299     ,partialdiff
300     ,pi
301     ,piece
302     ,piecewise
303     ,plus
304     ,power
305     ,primes
306     ,product
307     ,prsubset
308     ,quotient
309     ,rationals
310     ,real
311     ,reals
312     ,reln
313     ,rem
314     ,root
315     ,scalarproduct
316     ,sdev
317     ,sec
318     ,sech
319     ,selector
320     ,semantics
321     ,sep
322     ,set
323     ,setdiff
324     ,share
325     ,sin
326     ,sinh
327     ,subset
328     ,sum
329     ,tan
330     ,tanh
331     ,tendsto
332     ,times
333     ,transpose
334     ,true
335     ,union
336     ,uplimit
337     ,variance
338     ,vector
339     ,vectorproduct
340     ,xor
341 }
342
343 \prop_const_from_keyval:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
344 {
345     DocumentFragment = Art,
346     Aside = Note,
347     Title = H1,
348     Sub    = Span,
349     H7     = H6 ,

```

```

350     H8      = H6 ,
351     H9      = H6 ,
352     H10     = H6,
353     FENote= Note,
354     Em      = Span,
355     Strong= Span,
356 }

```

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

We fill the structure tags in to the seq. We allow all pdf1.7 and pdf2.0, and role map if needed the 2.0 tags.

```

357 % get tag name from number: \seq_item:Nn \g__tag_role_tags_seq { n }
358 % get tag number from name: \prop_item:Nn \g__tag_role_tags_prop { name }
359
360 \clist_map_inline:Nn \c__tag_role_sttags_pdf_pdfII_clist
361 {
362     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
363     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf2 }
364 }
365 \clist_map_inline:Nn \c__tag_role_sttags_only_pdf_clist
366 {
367     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
368     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf }
369 }
370 \clist_map_inline:Nn \c__tag_role_sttags_only_pdfII_clist
371 {
372     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
373     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf2 }
374 }
375 \pdf_version_compare:NnT > {1.9}
376 {
377     \clist_map_inline:Nn \c__tag_role_sttags_mathml_clist
378     {
379         \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
380         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ mathml }
381     }
382 }

```

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

```

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

```

1.4 Adding new tags and rolemapping

1.4.1 pdf 1.7 and earlier

With this versions only RoleMap is filled. At first the dictionary:

g__tag_role/RoleMap_dict

```
391 \pdfdict_new:n {g__tag_role/RoleMap_dict}
```

(End definition for g__tag_role/RoleMap_dict.)

__tag_role_add_tag:nn The pdf 1.7 version has only two arguments: new and rolemap name. To make pdf 2.0 types usable we directly define a rolemapping for them.

```
392 \cs_new_protected:Nn \__tag_role_add_tag:nn {(new) name, reference to old
```

```
393 {
```

```
394   \prop_if_in:NnF \g__tag_role_tags_prop {#1}
```

```
395   {
```

```
396     \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
```

```
397     {
```

```
398       \msg_info:nnn { tag }{new-tag}{#1}
```

```
399     }
```

```
400     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
```

```
401     \__tag_prop_gput:Nnx \g__tag_role_tags_prop { #1 }
```

```
402     {
```

```
403       \seq_count:N \g__tag_role_tags_seq
```

```
404     }
```

```
405     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ user }
```

```
406   }
```

```
407   \__tag_check_add_tag_role:nn {#1}{#2}
```

```
408   \tl_if_empty:nF { #2 }
```

```
409   {
```

```
410     \pdfdict_gput:nnx {g__tag_role/RoleMap_dict}
```

```
411     {#1}
```

```
412     {\pdf_name_from_unicode_e:n{#2}}
```

```
413   }
```

```
414 }
```

```
415 \cs_generate_variant:Nn \__tag_role_add_tag:nn {VV}
```

```
416
```

```
417 \pdf_version_compare:NnT < {2.0}
```

```
418 {
```

```
419   \prop_map_inline:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
```

```
420   {
```

```
421     \__tag_role_add_tag:nn {#1}{#2}
```

```
422   }
```

```
423 }
```

```
424
```

(End definition for __tag_role_add_tag:nn.)

1.4.2 The pdf 2.0 version

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

```
425 \cs_new_protected:Nn \__tag_role_add_tag:nnnn {tag/namespace/role/namespace
```

```
426 {
```

```
427   \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
```

```
428   {
```

```
429     \msg_info:nnn { tag }{new-tag}{#1}
```

```
430   }
```

```
431   \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
```

```
432   \__tag_prop_gput:Nnx \g__tag_role_tags_prop { #1 }
```

```

433     {
434         \seq_count:N \g__tag_role_tags_seq
435     }
436     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ #2 }
437     \__tag_check_add_tag_role:nn {#1}{#3}
438     \pdfdict_gput:nxx {g__tag_role/RoleMapNS_#2_dict}{#1}
439     {
440         [
441             \pdf_name_from_unicode_e:n{#3}
442             \c_space_tl
443             \pdf_object_ref:n {tag/NS/#4}
444         ]
445     }
446 }
447 \cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}

```

(End definition for __tag_role_add_tag:nnnn.)

1.5 Key-val user interface

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

```

tag
tag-namespace
role
role-namespace
add-new-tag
448 \keys_define:nn { __tag / tag-role }
449 {
450     ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
451     ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
452     ,role .tl_set:N = \l__tag_role_role_tmpa_tl
453     ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
454 }
455
456 \keys_define:nn { __tag / setup }
457 {
458     add-new-tag .code:n =
459     {
460         \keys_set_known:nnnN
461         {__tag/tag-role}
462         {
463             tag-namespace=user,
464             role-namespace=, %so that we can test for it.
465             #1
466             }{__tag/tag-role}\l_tmpa_tl
467         \tl_if_empty:NF \l_tmpa_tl
468         {
469             \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
470             \tl_set:Nx \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
471             \tl_set:Nx \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
472         }
473         \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
474         {
475             \prop_get:NVNTF
476             \g__tag_role_tags_NS_prop
477             \l__tag_role_role_tmpa_tl

```

```

478         \l__tag_role_role_namespace_tmpa_tl
479     {
480         \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
481         {
482             \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
483         }
484     }
485     {
486         \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
487     }
488 }
489 \pdf_version_compare:NnTF < {2.0}
490 {
491     %TODO add check for emptyness?
492     \__tag_role_add_tag:VV
493         \l__tag_role_tag_tmpa_tl
494         \l__tag_role_role_tmpa_tl
495 }
496 {
497     \__tag_role_add_tag:VVVV
498         \l__tag_role_tag_tmpa_tl
499         \l__tag_role_tag_namespace_tmpa_tl
500         \l__tag_role_role_tmpa_tl
501         \l__tag_role_role_namespace_tmpa_tl
502 }
503 }
504 }
505 \end{package}

```

(End definition for tag and others. These functions are documented on page 62.)

Part X

The tagpdf-space module

Code related to real space chars

Part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2021-07-03} {0.91}
4 {part of tagpdf - code related to real space chars}
5 </header>
```

1 Code for interword spaces

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

interwordspace

show-spaces

```
6 <*package>
7 \sys_if_engine_pdfTeX:T
8 {
9   \sys_if_output_pdf:TF
10   {
11     \pdfglyphtounicode{space}{0020}
12     \keys_define:nn { __tag / setup }
13     {
14       interwordspace .choices:nn = { true, on } { \pdfinterwordspaceon },
15       interwordspace .choices:nn = { false, off } { \pdfinterwordspaceon },
16       interwordspace .default:n = true,
17       show-spaces .bool_set:N = \l__tag_showspaces_bool
18     }
19   }
20   {
21     \keys_define:nn { __tag / setup }
22     {
23       interwordspace .choices:nn = { true, on, false, off }
24       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
25       interwordspace .default:n = true,
26       show-spaces .bool_set:N = \l__tag_showspaces_bool
27     }
28   }
29 }
30
31 \sys_if_engine_luaTeX:T
32 {
33   \keys_define:nn { __tag / setup }
34   {
35     interwordspace .choices:nn =
36     { true, on }
```



```

38         {
39             \bool_gset_true:N \g__tag_active_space_bool
40             \lua_now:e{ltx.__tag.func.markspaceon()}
41         },
42         interwordspace .choices:nn =
43             { false, off }
44         {
45             \bool_gset_false:N \g__tag_active_space_bool
46             \lua_now:e{ltx.__tag.func.markspaceoff()}
47         },
48         interwordspace .default:n = true,
49         show-spaces .choice:,
50         show-spaces / true .code:n =
51             {\lua_now:e{ltx.__tag.trace.showspace=true}},
52         show-spaces / false .code:n =
53             {\lua_now:e{ltx.__tag.trace.showspace=nil}},
54         show-spaces .default:n = true
55     }
56 }
57
58 \sys_if_engine_xetex:T
59 {
60     \keys_define:nn { __tag / setup }
61     {
62         interwordspace .choices:nn = { true, on }
63         { \msg_warning:nnn {tag}{sys-no-interwordspace}{xetex} },
64         interwordspace .choices:nn = { false, off }
65         { \msg_warning:nnn {tag}{sys-no-interwordspace}{xetex} },
66         interwordspace .default:n = true,
67         show-spaces .bool_set:N = \l__tag_showspace_bool
68     }
69 }

```

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

`__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{ltx.__tag.func.fakespace()}
76         \skip_horizontal:n{\c_zero_skip}
77         \group_end:
78     }
79 }
80 </package>

```

(End definition for `__tag_fakespace:.`)

Index

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

Symbols	
\\	10
_	174, 187
A	
activate	135
activate-all	151
activate-mc	151
activate-space	151
activate-struct	151
activate-tree	151
actualtext	42, 63, 144, 197, 269
actualtext-o	42, 63, 144, 197
actualtext-o _{uuu} lang	269
add-new-tag	448
\AddToHook	13, 16, 51, 166, 180
AF	63, 396
AFinline	63, 396
AFinline-o	63, 396
alttext	42, 63, 144, 197, 269
alttext-o	42, 63, 144, 197, 269
artifact	42, 144, 197
artifact-bool internal commands:	
__artifact-bool	102
artifact-type internal commands:	
__artifact-type	102
attr-unknown	33
attribute	64, 625
attribute-class	64, 591
B	
bool commands:	
\bool_gset_false:N	31, 45, 139, 189
\bool_gset_true:N	30, 39, 110, 159
\bool_if:NTF	9, 9, 12, 18, 23, 33, 69, 112, 126, 133, 157, 168, 169, 172, 176, 182, 185, 186, 212, 225, 230, 479
\bool_if:nTF	6
\bool_lazy_all:nTF	46
\bool_lazy_and:nnTF	63, 73
\bool_lazy_and_p:nn	8
\bool_new:N	11, 15, 16, 29, 55, 80, 81, 82, 83, 84, 86, 88, 157, 158
\bool_set_false:N	147, 148, 160, 190, 194
\bool_set_true:N	85, 87, 193
C	
\c	134, 135
c@g internal commands:	
\c@g__tag_MCID_abs_int	9, 25, 30, 45, 59, 110, 128, 162
\c@g__tag_parenttree_obj_int	52
\c@g__tag_struct_abs_int	6, 46, 99, 102, 104, 280, 286, 299, 311, 323, 335, 347, 359, 366, 379, 391, 403, 414, 418, 419, 422, 424, 430, 434, 435, 438, 440, 449, 450, 451, 452, 455, 458, 462, 475, 477, 483, 618, 621, 663
clist commands:	
\clist_const:Nn	64, 77, 78, 110, 132, 148
\clist_map_inline:Nn	360, 365, 370, 377
\clist_map_inline:nn	373
\clist_new:N	75
\clist_set:Nn	595, 629
color commands:	
\color_select:n	174, 187
cs commands:	
\cs_generate_variant:Nn	22, 76, 87, 90, 91, 92, 93, 94, 95, 96, 97, 97, 104, 112, 119, 124, 128, 138, 139, 140, 140, 141, 142, 143, 152, 415, 447, 564
\cs_gset_eq:NN	179
\cs_if_exist:NTF	53
\cs_if_exist_p:N	9
\cs_if_free:NTF	39
\cs_new:Nn	21, 68, 73, 99, 103, 121, 126, 130
\cs_new:Npn	9, 43, 55, 56, 61, 120, 125, 196, 261, 565
\cs_new_protected:Nn	72, 134, 153, 184, 392, 425, 504, 530
\cs_new_protected:Npn	15, 23, 25, 29, 32, 35, 44, 50, 57, 60, 63, 66, 66, 69, 71, 77, 81, 82, 88, 89, 89, 92, 96, 96, 103, 104, 105, 109, 113, 113, 118, 119, 130, 131, 135, 138, 141, 144, 146, 147, 153, 154, 165, 176, 181, 187, 188, 200, 210, 210, 227, 444, 555, 578
\cs_set:Npn	38, 43
\cs_set_eq:NN	19, 20, 21, 46, 47, 48, 129, 130, 131, 132, 133, 134, 135, 136, 150

<code>\cs_set_protected:Npn</code>	9, 16, 23, 30, 49, 56
<code>\cs_to_str:N</code> 12, 19, 26, 33, 52, 53, 59, 60	
D	
<code>\DeclareDocumentMetadata</code>	21
<code>\DeclareOption</code>	30, 31
default commands:	
<code>default_fontid</code>	432
<code>default_space_char</code>	432
<code>\documentclass</code>	22
E	
E	63, 269
<code>\ExecuteOptions</code>	32
exp commands:	
<code>\exp_args:Ne</code>	263, 453
<code>\exp_args:Nee</code>	57
<code>\exp_args:NNno</code>	469
<code>\exp_args:NNnx</code>	39
<code>\exp_args:NNx</code>	39
<code>\exp_args:Nnx</code> .. 60, 198, 249, 309, 313	
<code>\exp_args:Nv</code>	101, 123, 172
<code>\exp_args:Nx</code>	216
<code>\exp_not:n</code>	64
F	
fi commands:	
<code>\fi:</code>	19
file commands:	
<code>\file_input:n</code>	182
<code>\fontencoding</code>	6
<code>\fontfamily</code>	6
<code>\fontseries</code>	6
<code>\fontshape</code>	6
<code>\fontsize</code>	6
G	
group commands:	
<code>\group_begin:</code>	74, 108, 146, 157, 413, 429, 448
<code>\group_end:</code>	77, 131, 150, 181, 425, 441, 499
H	
hook commands:	
<code>\hook_gput_code:nnn</code>	7, 7, 21, 26, 50, 53, 137, 138, 195, 208, 218, 223, 227, 231
<code>\hook_use:n</code>	214
I	
if commands:	
<code>\if_mode_horizontal:</code>	19
<code>\ignorespaces</code>	24, 13
int commands:	
<code>\int_case:nnTF</code>	143
<code>\int_compare:nNnTF</code> 60, 73, 98, 128, 155, 158, 183, 189, 194, 396, 427	
<code>\int_compare:nTF</code>	77, 223, 611, 613, 615, 633, 659
<code>\int_eval:n</code>	47, 64, 88, 232, 280, 286, 299, 311, 323, 335, 347, 359, 366, 379, 391, 403, 450, 451, 452, 455, 458, 462, 483, 618, 621, 663
<code>\int_gincr:N</code>	25, 162, 168, 449
<code>\int_gset:Nn</code>	43, 55
<code>\int_gzero:N</code>	8, 51
<code>\int_new:N</code>	10, 76, 79, 159
<code>\int_rand:n</code> .. 43, 44, 46, 48, 50, 52, 53	
<code>\int_set:Nn</code> 161, 162, 163, 164, 165	
<code>\int_step_inline:nnnn</code>	46, 71, 74, 91, 208, 214, 383
<code>\int_to_Hex:n</code> 43, 44, 46, 48, 50, 52, 53	
<code>\int_use:N</code>	9, 30, 44, 45, 59, 99, 102, 104, 108, 110, 112, 126, 128, 174, 187, 414, 418, 419, 422, 424, 430, 434, 435, 438, 440, 565
intarray commands:	
<code>\intarray_gset:Nnn</code>	186
<code>\intarray_item:Nn</code>	188, 191
<code>\intarray_new:Nn</code>	178
interwordspace	6
iow commands:	
<code>\iow_newline:</code>	171
<code>\iow_now:Nn</code>	39
K	
keys commands:	
<code>\keys_define:nn</code> 12, 21, 34, 54, 60, 66, 102, 128, 140, 144, 151, 160, 197, 269, 396, 448, 456, 584, 591, 625	
<code>\keys_set:nn</code>	9, 51, 111, 144, 163, 199, 310, 314, 461
<code>\keys_set_known:nnnN</code>	460
L	
label	42, 63, 144, 197, 269
lang	63
legacy commands:	
<code>\legacy_if:nTF</code>	37
<code>\llap</code>	174
log	160
ltx. internal commands:	
<code>ltx.__tag.func.fakespace</code>	351
<code>ltx.__tag.func.fill_parent_tree_-line</code>	692
<code>ltx.__tag.func.get_num_from</code>	258
<code>ltx.__tag.func.get_tag_from</code>	277

\pdfglyphtounicode	11	\rlap	187
\pdfinterwordspaceon	14, 15	role	448
pdfmanagement commands:		role-missing	34
\pdfmanagement_add:nnn	25, 26, 170, 172, 174, 229	role-namespace	448
\pdfmanagement_if_active_p:	9, 10	role-tag	37
\pdfmanagement_remove:nn	176	role-unknown	34
prg commands:		role-unknown-tag	34
\prg_do_nothing:	179		
\prg_generate_conditional_-variant:Nnn	89	S	
\prg_new_conditional:Nnn	10, 58	\selectfont	6
\prg_new_conditional:Npnn	44, 61, 71	seq commands:	
\prg_new_eq_conditional:Nnn	17, 72	\seq_clear:N	213
\prg_return_false:	14, 58, 68, 68, 78	\seq_const_from_clist:Nn	16, 28
\prg_return_true:	13, 55, 65, 69, 75	\seq_count:N	145, 383, 403, 434, 611, 613, 615, 633, 659
\ProcessOptions	33	\seq_get:NNTF	233, 469, 517, 524
prop commands:		\seq_gpop:NN	510
\prop_clear:N	73	\seq_gpop:NNTF	86, 511
\prop_const_from_keyval:Nn	343	\seq_gpop_left:NN	132
\prop_count:N	94	\seq_gpush:Nn	11, 13, 69, 76, 475, 476
\prop_get:NnNTF	35, 96, 111, 126, 283, 475	\seq_gput_left:Nn	137, 603
\prop_gput:Nnn	25, 27, 39, 83, 93, 131, 146, 363, 368, 373, 380, 405, 436, 580, 648	\seq_gput_right:Nn	32, 132, 133, 233
\prop_if_exist:NTF	25, 534	\seq_gremove_duplicates:N	157
\prop_if_in:NnNTF	58, 83, 91, 171, 201, 394, 480, 599, 637, 641	\seq_item:Nn	133, 164, 276, 277, 357, 387, 470, 471
\prop_item:Nn	32, 62, 83, 134, 162, 205, 266, 275, 358, 646, 653	\seq_log:N	131, 162
\prop_map_inline:Nn	189, 419	\seq_map_inline:Nn	191, 597, 635
\prop_map_tokens:Nn	207	\seq_new:N	10, 12, 13, 18, 73, 74, 130, 575
\prop_new:N	9, 10, 11, 72, 129, 574, 577	\seq_set_from_clist:NN	596, 630
\prop_put:Nnn	80, 94	\seq_set_map:Nnn	158, 605
\prop_show:N	58, 136, 494, 497, 621, 642	\seq_set_split:Nnn	96, 275, 469
\ProvidesExplFile	3	\seq_show:N	51, 132, 135, 243, 478, 495, 498, 507
\ProvidesExplPackage	3, 3, 3, 3, 3, 3, 3, 3, 3, 570	\seq_use:Nn	169, 176, 612
		\l_tmpa_seq	213, 233, 243, 469, 470, 471
R		shipout commands:	
raw	42, 144, 197	\g_shipout_readonly_int	44, 108, 126, 232
ref	63, 269	show-spaces	6
ref commands:		\ShowTagging	13, 25, 49
\ref_attribute_gset:nnnn	98, 100, 107, 109, 111	skip commands:	
\ref_label:nn	95, 116	\skip_horizontal:n	76
\ref_value:nn	376	\c_zero_skip	76
\ref_value:nnn	7, 53, 53, 55, 122, 127	stash	42, 63, 102, 269
ref internal commands:		\stepcounter	258
__ref_value:nnn	58, 61	str commands:	
regex commands:		\str_const:Nn	41
\regex_replace_once:nnN	133	\str_new:N	71
\RequirePackage	20, 34, 188, 191	\str_set_convert:Nnnn	97, 157, 167, 177, 187, 218, 235, 252, 269, 293, 305, 317, 329, 341, 353, 385
		\str_use:N	229, 246, 263, 282
		\l_tmpa_str	25, 26, 31
		\string	20, 21, 22

struct-faulty-nesting	23	\g__tag_active_mc_bool	33, 49, 63, 80, 154
struct-label-unknown	29	\l__tag_active_mc_bool	52, 63, 84, 148
struct-missing-tag	26	\g__tag_active_space_bool	9, 39, 45, 80, 153
struct-no-objnum	22	\g__tag_active_struct_bool	48, 73, 80, 156, 230
struct-show-closing	31	\l__tag_active_struct_bool	51, 73, 84, 147
struct-stack	25, 128	\g__tag_active_tree_bool	9, 23, 50, 80, 155, 212, 225
struct-used-twice	27	__tag_add_document_structure:n	135, 135, 146
sys commands:		\g__tag_attr_class_used_seq	157, 158, 573, 603
\sif_engine luatex:TF	30, 32, 46, 47, 58, 70, 71, 150, 180	\g__tag_attr_entries_prop	163, 573, 580, 599, 637, 642, 646
\sif_engine pdftex:TF	7, 48	__tag_attr_new_entry:nn	578, 578, 588
\sif_engine xetex:TF	58	\g__tag_attr_objref_prop	573, 641, 648, 653
\sif_output_pdf:TF	9, 11	\l__tag_attr_value_tl	573, 631, 650, 655, 657, 661, 665
sys-no-interwordspace	41	__tag_check_add_tag_role:nn	119, 119, 407, 437
T			
tabsorder	168	__tag_check_if_active_mc:	61
tag	42, 62, 144, 197, 269, 448	__tag_check_if_active_mc:TF	61, 65, 84, 106, 136, 155, 186
tag commands:		__tag_check_if_active_struct:	71
\tag_get:n	13, 72, 43, 43, 69, 72	__tag_check_if_active_struct:TF	31, 61, 446, 508, 532, 558
\tag_if_active:	44	__tag_check_info_closing-	
\tag_if_active:TF	13, 44	struct:n	96, 96, 104, 513
\tag_if_active_p:	13, 44	__tag_check_init_mc_used:	176, 176, 179, 185
\tag_mc_artifact_group_begin:n	41, 50, 50	__tag_check_mc_if_nested:	109, 138, 138, 158
\tag_mc_artifact_group_end:	41, 50, 57	__tag_check_mc_if_open:	138, 138, 146, 188
\tag_mc_begin:n	8, 41, 13, 53, 94, 104, 104, 153, 153, 173, 177, 186, 201, 224	__tag_check_mc_pushed_popped:nn	70, 77, 90, 93, 98, 153, 153
\tag_mc_begin_pop:n	41, 61, 63, 82, 215, 238	__tag_check_mc_tag:N	117, 165, 165, 165
\tag_mc_end:	41, 20, 60, 73, 104, 134, 175, 184, 184, 188, 213, 236	__tag_check_mc_used:n	79, 132, 181, 181
\tag_mc_end_push:	41, 52, 63, 63, 199, 222	\g__tag_check_mc_used_intarray	176, 186, 188, 191
\tag_mc_if_in:	17, 58, 72	__tag_check_no_open_struct:	105, 105, 515, 522
\tag_mc_if_in:TF	41, 10, 30	__tag_check_show_MCID_by_page:	200, 200
\tag_mc_if_in_p:	41, 10	__tag_check_struct_used:n	109, 109, 537
\tag_mc_use:n	41, 25, 29, 29	__tag_check_structure_has_tag:n	81, 81, 462
\tag_stop_group_begin:	54, 144, 144		
\tag_stop_group_end:	59, 144, 150		
\tag_struct_begin:n	62, 34, 171, 200, 223, 444, 444		
\tag_struct_end:	62, 39, 190, 214, 237, 444, 504		
\tag_struct_insert_annot:nn	62, 78, 212, 235, 555, 555, 564		
\tag_struct_parent_int:	62, 78, 205, 212, 228, 235, 555, 565		
\tag_struct_use:n	62, 44, 530, 530		
tag internal commands:			
__tag_activate_mark_space	415		

```

\__tag_check_structure_tag:N ...
..... 89, 89, 278
__tag_fakespace ..... 351
\__tag_fakespace: ..... 70, 72, 154
\__tag_finish_structure: .....
..... 13, 16, 210, 210
\__tag_get_data_mc_tag: .....
..... 103, 103, 196, 196
\__tag_get_data_struct_tag: 261, 261
__tag_get_mathsubtype ..... 250
\__tag_get_mc_abs_cnt: 9, 9, 19, 20,
60, 90, 92, 101, 142, 150, 169, 178,
205, 213, 229, 246, 263, 280, 293, 303
__tag_get_mc_cnt_type_tag ..... 244
__tag_get_num_from ..... 258
__tag_get_tag_from ..... 277
\g__tag_in_mc_bool .....
..... 11, 12, 18, 110, 139, 159, 189
__tag_insert_bdc_node ..... 329
__tag_insert_bmc_node ..... 322
__tag_insert_emc_node ..... 315
\__tag_lastpagelabel: .... 35, 35, 52
__tag_log ..... 172
\l__tag_loglevel_int .....
..... 79, 98, 128, 156, 159,
161, 162, 163, 164, 165, 183, 396, 427
__tag_mark_spaces ..... 356
\l__tag_mc_artifact_bool .....
..... 15, 105, 112, 160, 168, 190
\l__tag_mc_artifact_type_tl . 14,
109, 113, 114, 117, 121, 125, 129, 316
\__tag_mc_bdc:nn ... 18, 21, 22, 62, 94
\__tag_mc_bdc_mcid:n ..... 23, 66
\__tag_mc_bdc_mcid:nn . 23, 23, 68, 73
\__tag_mc_bmc:n ..... 18, 19, 90
\__tag_mc_bmc_artifact: .. 88, 88, 100
\__tag_mc_bmc_artifact:n . 88, 92, 101
\__tag_mc_emc: ..... 18, 20, 141
\__tag_mc_handle_artifact:N ....
..... 88, 96, 114
\__tag_mc_handle_mc_label:n ....
..... 21, 21, 124, 173
\__tag_mc_handle_mcid:nn .....
..... 23, 71, 76, 118
\__tag_mc_handle_stash:n .... 41,
77, 77, 87, 128, 130, 130, 152, 178
\__tag_mc_if_in: ... 10, 17, 58, 58, 72
\__tag_mc_if_in:TF .. 10, 67, 140, 148
\__tag_mc_if_in_p: ..... 10
\__tag_mc_insert_mcid_kids:n ...
..... 121, 121, 134, 137
\__tag_mc_insert_mcid_single_
kids:n ..... 121, 126, 135
\l__tag_mc_key_label_tl .....
..... 17, 121, 124, 170, 173, 195, 288
\l__tag_mc_key_properties_tl ...
..... 17, 120, 153, 161, 162, 163,
172, 173, 182, 183, 192, 193, 210,
223, 224, 240, 241, 257, 258, 274, 275
\l__tag_mc_key_stash_bool .....
..... 15, 104, 126, 176
\g__tag_mc_key_tag_tl .....
.. 17, 19, 103, 140, 149, 193, 196, 202
\l__tag_mc_key_tag_tl .....
. 17, 117, 119, 148, 165, 167, 192, 201
\__tag_mc_lua_set_mc_type_attr:n
..... 73, 73, 97, 167
\__tag_mc_lua_unset_mc_type_
attr: ..... 73, 99, 191
\g__tag_mc_parenttree_prop .....
..... 12, 13, 83, 83, 147
\l__tag_mc_ref_abspage_tl .....
..... 8, 26, 38, 46, 54
\__tag_mc_set_label_used:n 25, 25, 42
\g__tag_mc_stack_seq 13, 69, 76, 86, 162
\l__tag_mc_tmpa_tl ..... 9, 40, 43, 47
g__tag_MCID_abs_int ..... 7
\g__tag_MCID_byabspage_prop ....
..... 6, 36, 45, 53
\g__tag_MCID_tmp_bypage_int ....
..... 10, 43, 51, 64, 112
\g__tag_mode_lua_bool .....
..... 29, 30, 31, 69, 133, 157, 186
\__tag_new_output_prop_handler:n
..... 56, 66, 75, 451
__tag_pairs_prop ..... 189
\l__tag_para_bool .....
..... 157, 162, 169, 182, 193, 194
\g__tag_para_int ... 157, 168, 174, 187
\l__tag_para_show_bool .....
..... 157, 163, 172, 185
\__tag_parenttree_add_objr:nn ...
..... 60, 60, 250
\l__tag_parenttree_content_tl ...
..... 67, 86, 98, 112, 120, 140, 143
\g__tag_parenttree_objr_tl 59, 62, 140
__tag_pdf_object_ref ..... 336
\__tag_prop_gput:Nnn .....
..... 9, 23, 34, 38, 44, 52,
78, 85, 129, 131, 138, 162, 173, 180,
279, 285, 298, 310, 322, 334, 346,
358, 365, 378, 385, 390, 401, 402,
421, 432, 437, 457, 482, 543, 617, 662
\__tag_prop_item:Nn .. 9, 43, 129, 134
\__tag_prop_new:N ..... 7,
8, 9, 9, 12, 74, 129, 129, 140, 450
\__tag_prop_show:N 9, 56, 129, 136, 143

```

__tag_ref_label:nn	23, 57, 113 , 113, 119, 466
__tag_ref_value:nnn	28, 33, 78, 82, 98, 99, 112,
	120 , 120, 124, 225, 236, 535, 541, 544
__tag_ref_value_lastpage:nn ...	57, 71, 74, 125 , 125, 204, 218
\c__tag_refmc_clist	77
\c__tag_refstruct_clist	77
g__tag_role/RoleMap_dict	391
__tag_role_add_tag:nn	392, 392, 415, 421, 492
__tag_role_add_tag:nnnn	425, 425, 447, 497
__tag_role_NS_new:nnn	101, 15 , 15, 57, 58, 59, 61
\g__tag_role_NS_prop	10 , 39, 189, 207, 283, 480
\l__tag_role_role_namespace_-	
tmpa_tl	11 ,
	453, 473, 478, 480, 482, 486, 501
\l__tag_role_role_tmpa_tl	11 , 452, 471, 477, 494, 500
\c__tag_role_sttags_mathml_clist	
clist	63 , 377
\c__tag_role_sttags_only_pdf_-	
clist	63 , 365
\c__tag_role_sttags_only_pdfII_-	
clist	63 , 370
\c__tag_role_sttags_pdf_pdfII_-	
clist	63 , 360
\c__tag_role_sttags_pdfII_to_-	
pdf_prop	63 , 419
\l__tag_role_tag_namespace_tmpa_-	
tl	11 , 451, 499
\l__tag_role_tag_tmpa_tl	11 , 450, 470, 493, 498
\g__tag_role_tags_NS_prop	9 , 171,
	275, 363, 368, 373, 380, 405, 436, 476
\g__tag_role_tags_prop	6 , 91, 126, 358, 385, 394, 401, 432
\g__tag_role_tags_seq	6 , 357, 362, 367,
	372, 379, 383, 387, 400, 403, 431, 434
\c__tag_role_userNS_id_str	100, 41 , 61
__tag_seq_gput_right:Nn	9 , 30, 91, 105, 121, 129 ,
	132, 139, 362, 367, 372, 379, 400, 431
__tag_seq_item:Nn ...	9 , 38, 129 , 133
__tag_seq_new:N	7 , 9 , 9 , 16, 76, 129 , 130, 141, 452
__tag_seq_show:N .	9 , 49, 129 , 135, 142
__tag_show_spacemark	342
\l__tag_showspaces_bool ...	17, 26, 67
__tag_space_chars_shipout	436
g__tag_struct_0_prop	74
\l__tag_struct_elem_stash_bool ..	55, 272, 480
__tag_struct_exchange_kid_-	
command:N	130 , 130, 140, 159
__tag_struct_fill_kid_key:n ...	141 , 141, 214
__tag_struct_get_dict_content:nN	
.....	188 , 188, 215
__tag_struct_insert_annot:nn ...	227, 227, 560
\l__tag_struct_key_label_tl	54 , 271, 464, 466
__tag_struct_kid_mc_gput_-	
right:nn	80, 89 , 89
__tag_struct_kid_OBJR_gput_-	
right:nn	113 , 113, 128, 241
__tag_struct_kid_struct_gput_-	
right:nn ...	103 , 103, 112, 491, 539
g__tag_struct_kids_0_seq	74
\g__tag_struct_objR_seq	9
__tag_struct_output_prop_aux:nn	
.....	56 , 56, 70
\g__tag_struct_stack_current_tl .	
.....	14 ,
	73, 81, 85, 135, 143, 149, 266, 477,
	489, 493, 494, 497, 513, 519, 540, 547
\l__tag_struct_stack_parent_-	
tmpa_tl	14 , 235,
	243, 255, 471, 486, 490, 492, 495, 498
\g__tag_struct_stack_seq	10 , 234, 470, 475, 478, 507, 511, 517
\c__tag_struct_StructElem_-	
entries_seq	16
\c__tag_struct_StructTreeRoot_-	
entries_seq	16
\g__tag_struct_tag_NS_tl	52 , 277, 283
\g__tag_struct_tag_stack_seq ...	12 , 131, 132, 476, 510, 524
\g__tag_struct_tag_tl	52 , 276, 278, 282, 476, 526
__tag_struct_write_obj:n	42, 48, 210 , 210
\g__tag_tagunmarked_bool	88 , 166
\l__tag_tmpa_clist	70 , 595, 596, 629, 630
\l__tag_tmpa_int	70
\l__tag_tmpa_prop ..	70 , 73, 81, 94, 96
\l__tag_tmpa_seq	70 ,
	158, 170, 275, 276, 277, 596, 597,
	605, 611, 613, 615, 630, 633, 635, 659

<code>\l__tag_tmpa_str</code>	<code>\tagpdfsetup</code>
..... 70 , 158 , 163 , 168 , 173 , 178 ,	24 , 64 , 6
183 , 188 , 193 , 219 , 224 , 229 , 236 ,	<code>tagstruct</code>
241 , 246 , 253 , 258 , 263 , 270 , 275 ,	98
282 , 294 , 301 , 306 , 313 , 318 , 325 ,	<code>\tagstructbegin</code>
330 , 337 , 342 , 349 , 354 , 361 , 386 , 393	24 , 32 , 137
<code>\l__tag_tmpa_tl</code>	<code>\tagstructend</code>
.... 33 , 34 , 41 , 70 , 76 , 83 , 86 , 88 ,	24 , 32 , 138
93 , 94 , 96 , 97 , 100 , 102 , 132 , 136 ,	<code>tagstructobj</code>
137 , 156 , 167 , 174 , 179 , 202 , 210 ,	98
215 , 220 , 283 , 288 , 372 , 375 , 381 ,	<code>\tagstructuse</code>
510 , 511 , 517 , 519 , 524 , 526 , 609 , 620	24 , 32
<code>\l__tag_tmpb_seq</code>	<code>tagunmarked</code>
70 , 605 , 612	166
<code>__tag_tree_fill_parenttree:</code> ...	TeX and L ^A T _E X 2 _ε commands:
..... 68 , 69 , 138	<code>\@auxout</code>
<code>__tag_tree_lua_fill_parenttree:</code>	39
..... 118 , 118 , 135	<code>\@bsphack</code>
<code>__tag_tree_write_classmap:</code> ...	115
..... 154 , 154 , 217	<code>\@esphack</code>
<code>__tag_tree_write_namespaces:</code> ...	117
..... 187 , 187 , 218	<code>\@gobble</code>
<code>__tag_tree_write_parenttree:</code> ...	24 , 48
..... 131 , 131 , 215	<code>\@secondoftwo</code>
<code>__tag_tree_write_rolemap:</code>	24 , 48
..... 147 , 147 , 216	<code>\tiny</code>
<code>__tag_tree_write_structelements:</code>	174 , 187
..... 44 , 44 , 219	<code>title</code>
<code>__tag_tree_write_structtreeroot:</code>	63 , 269
..... 32 , 32 , 220	<code>title-o</code>
<code>tag-namespace</code>	63 , 269
448	tl commands:
<code>tag/struct/0</code> internal commands:	<code>\c_space_tl</code> 62 , 64 , 88 , 89 , 95 , 97 , 99 ,
<code>__tag/struct/0</code>	104 , 143 , 160 , 181 , 205 , 442 , 612 , 652
20	<code>\tl_clear:N</code>
<code>tag/tree/namespace</code> internal commands:	156 , 161 , 190 , 372
<code>__tag/tree/namespace</code>	<code>\tl_gput_right:Nn</code>
186	62
<code>tag/tree/parenttree</code> internal commands:	<code>\tl_gset:Nn</code>
<code>__tag/tree/parenttree</code>	73 , 140 ,
51	149 , 193 , 202 , 276 , 277 , 477 , 519 , 526
<code>tag/tree/rolemap</code> internal commands:	<code>\tl_if_empty:N</code>
<code>__tag/tree/rolemap</code>	26 ,
146	34 , 99 , 121 , 167 , 170 , 173 , 463 , 467 , 473
<code>tagabspage</code>	<code>\tl_if_empty:N</code>
98	34 , 121 , 408
<code>tagmcabs</code>	<code>\tl_if_eq:NnTF</code>
98	88
<code>\tagmcbegin</code>	<code>\tl_if_exist:N</code>
24 , 11	63
<code>\tagmcend</code>	<code>\tl_new:N</code>
24 , 11	8 ,
<code>tagmcid</code>	9 , 11 , 12 , 13 , 14 , 14 , 14 , 15 , 17 , 18 ,
98	19 , 20 , 27 , 52 , 53 , 54 , 59 , 67 , 70 , 576
<code>\tagmcifin</code>	<code>\tl_put_right:Nn</code>
24	86 , 98 , 111 , 140 ,
<code>\tagmcifinTF</code>	153 , 162 , 163 , 172 , 173 , 182 , 183 ,
28	192 , 193 , 198 , 210 , 223 , 224 , 240 ,
<code>\tagmcuse</code>	241 , 257 , 258 , 274 , 275 , 375 , 650 , 657
24 , 11	<code>\tl_set:Nn</code>
<code>\tagpdfifluatexT</code> 26 , 33 , 76 , 109 , 113 , 117 , 120 ,
46	121 , 125 , 129 , 148 , 167 , 192 , 201 ,
<code>\tagpdfifluatexTF</code>	202 , 288 , 470 , 471 , 482 , 486 , 609 , 631
46	<code>\tl_show:N</code>
<code>\tagpdfifpdfTeX</code>	489 , 490 , 655 , 661
48	<code>\tl_tail:n</code>
<code>\tagpdfifpdfTeXTF</code>	264
46	<code>\tl_to_str:n</code>
<code>\tagpdfparaOff</code>	27 , 39 , 59
26 , 193	<code>\tl_use:N</code>
<code>\tagpdfparaOn</code>	64
26 , 193	<code>\l_tmpa_tl</code>
	114 , 126 , 466 , 467 , 469
	token commands:
	<code>\token_to_str:N</code>
	41
	<code>tree-mcid-index-wrong</code>
	39
	U
	<code>\unskip</code>
	24 , 19
	use commands:
	<code>\use:N</code>
	43
	<code>\use_ii:nn</code>
	207