

# The **tagpdf-tree** module

## Commands trees and main dictionaries

### part of the tagpdf package

Ulrike Fischer\*

Version 0.82, released 2021-06-14

```
1 <@@=tag>
2 <*tree>
3 \ProvidesExplPackage {tagpdf-tree-code} {2021-06-14} {0.82}
4 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
```

## 1 Trees 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

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

### 1.1 Structure tree

The following commands are needed to write out the structure.

`__tag/struct/0` This is the object for the root object, the StructTreeRoot

```
18 \pdf_object_new:nn { __tag/struct/0 }{ dict }
```

*(End definition for \_\_tag/struct/0.)*

The StructTreeRoot must be added to the catalog. TODO Should this be done later to prevent other packages to interfere?

```
19 \hook_gput_code:nnn{begindocument}{tagpdf}
20 {
```

---

\*E-mail: [fischer@troubleshooting-tex.de](mailto:fischer@troubleshooting-tex.de)

```

21 \bool_if:NT \g__tag_active_struct_bool
22 {
23     \pdfmanagement_add:nnx
24     { Catalog }
25     { StructTreeRoot }
26     { \pdf_object_ref:n { __tag/struct/0 } }
27 }
28 }

```

`\__tag_tree_write_structtreeroot:` This writes out the root object.

```

29 \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
30 {
31     \__tag_struct_write_obj:n { 0 }
32 }

```

*(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

```

33 \cs_new_protected:Npn \__tag_tree_write_structelements:
34 {
35     \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
36     {
37         \__tag_struct_write_obj:n { ##1 }
38     }
39 }

```

*(End definition for \\_\_tag\_tree\_write\_structelements:.)*

## 1.2 ParentTree

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

```

40 \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`.

```

41 \newcounter { g__tag_parenttree_obj_int }
42 \hook_gput_code:nnn{begindocument}{tagpdf}
43 {
44     \int_gset:Nn
45     \c@g__tag_parenttree_obj_int
46     { \__tag_ref_value_lastpage:nn{abspage}{100} }
47 }

```

*(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

```
48 \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 only for objects like annotations, pages are handled elsewhere.

```
49 \cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 %#1 Structparent number, #2 objref
50 {
51   \tl_gput_right:Nx \g__tag_parenttree_objr_tl
52   {
53     #1 \c_space_tl #2 ^^J
54   }
55 }
```

(End definition for \\_\_tag\_parenttree\_add\_objr:nn.)

\l\_\_tag\_parenttree\_content\_tl

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

```
56 \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.

```
57 \cs_new_protected:Npn \__tag_tree_fill_parenttree:
58 {
59   \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{abspage}{-1}} %not quite clear i
60   { %page ##1
61     \prop_clear:N \l__tag_tmpa_prop
62     \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{tagmcabs}{-1}}
63     {
64       %mcid###1
65       \int_compare:nT
66       {\__tag_ref_value:enn{mcid-###1}{tagabspace}{-1}=##1} %mcid is on current page
67       {% yes
68         \prop_put:Nxx
69         \l__tag_tmpa_prop
70         {\__tag_ref_value:enn{mcid-###1}{tagmcid}{-1}}
71         {\prop_item:Nn \g__tag_mc_parenttree_prop {###1}}
72       }
73     }
74     \tl_put_right:Nx\l__tag_parenttree_content_tl
75     {
76       \int_eval:n {##1-1}\c_space_tl
77       [\c_space_tl %]
78     }
79     \int_step_inline:nnnn
80     {0}
81     {1}
82     { \prop_map_function:NN \l__tag_tmpa_prop\__tag_prop_count:nn -1 }
83     {
84       \prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
85       {% page#1:mcid##1:\l__tag_tmpa_tl :content
86       \tl_put_right:Nx \l__tag_parenttree_content_tl
```

```

87         {
88             \prop_item:cn { g__tag_struct_\l__tag_tmpa_tl _prop } {objref}
89             \c_space_tl
90         }
91     }
92     {
93         \msg_warning:nn { tag } {tree-mcid-index-wrong}
94     }
95 }
96 \tl_put_right:Nn
97   \l__tag_parenttree_content_tl
98   {%[
99   ]^^J
100 }
101 }
102 }

```

(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.

```

103 \cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
104 {
105     \tl_set:Nn \l__tag_parenttree_content_tl
106     {
107         \directlua
108         {
109             ltx.__tag.func.output_parenttree
110             (
111                 \int_use:N\g_shipout_readonly_int
112             )
113         }
114     }
115 }

```

(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?

```

116 \cs_new_protected:Npn \__tag_tree_write_parenttree:
117 {
118     \bool_if:NTF \g__tag_mode_lua_bool
119     {
120         \__tag_tree_lua_fill_parenttree:
121     }
122     {
123         \__tag_tree_fill_parenttree:
124     }
125     \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
126     \pdf_object_write:nx { __tag/tree/parenttree }
127     {
128         /Nums\c_space_tl [\l__tag_parenttree_content_tl]
129     }
130 }

```

(End definition for `\__tag_tree_write_parenttree:`)

### 1.3 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.  
131 \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.

```
132 \cs_new_protected:Npn \__tag_tree_write_rolemap:  
133 {  
134   \pdf_object_write:nx { __tag/tree/rolemap }  
135   {  
136     \pdfdict_use:n{g__tag_role/RoleMap_dict}  
137   }  
138 }  
(End definition for \__tag_tree_write_rolemap:.)
```

### 1.4 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:  
139 \cs_new_protected:Npn \__tag_tree_write_classmap:  
140 {  
141   \tl_gclear:N \g__tag_attr_class_content_tl  
142   \seq_gremove_duplicates:N \g__tag_attr_class_used_seq  
143   \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq  
144   {  
145     /##1\c_space_tl  
146     <<  
147     \prop_item:Nn  
148     \g__tag_attr_entries_prop  
149     {##1}  
150     >>  
151   }  
152   \tl_gset:Nx \g__tag_attr_class_content_tl  
153   {  
154     \seq_use:Nn  
155     \l__tag_tmpa_seq  
156     { \iow_newline: }  
157   }  
158   \tl_if_empty:NF  
159   \g__tag_attr_class_content_tl  
160   {  
161     \pdf_object_new:nn { __tag/tree/classmap }{ dict }  
162     \pdf_object_write:nx  
163     { __tag/tree/classmap }  
164     { \g__tag_attr_class_content_tl }
```

```

165     \__tag_prop_gput:cnx
166     { g__tag_struct_0_prop }
167     { ClassMap }
168     { \pdf_object_ref:n { __tag/tree/classmap } }
169   }
170 }

```

(End definition for \\_\_tag\_tree\_write\_classmap:.)

## 1.5 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

```

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

```

(End definition for \_\_tag/tree/namespaces.)

\\_\_tag\_tree\_write\_namespaces:

```

172 \cs_new_protected:Npn \__tag_tree_write_namespaces:
173 {
174   \prop_map_inline:Nn \g__tag_role_NS_prop
175   {
176     \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
177     {
178       \pdf_object_write:nx {__tag/RoleMapNS/##1}
179       {
180         \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
181       }
182       \pdfdict_gput:nnx{g__tag_role/namespace_##1_dict}
183       {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
184     }
185     \pdf_object_write:nx{tag/NS/##1}
186     {
187       \pdfdict_use:n {g__tag_role/namespace_##1_dict}
188     }
189   }
190   \pdf_object_write:nx {__tag/tree/namespaces}
191   {
192     \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
193   }
194 }

```

(End definition for \\_\_tag\_tree\_write\_namespaces:.)

## 1.6 Finishing the structure

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

\\_\_tag\_finish\_structure:

```

195 \cs_new_protected:Npn \__tag_finish_structure:
196 {
197   \__tag_tree_write_parenttree:

```

```

198     \__tag_tree_write_rolemap:
199     \__tag_tree_write_classmap:
200     \__tag_tree_write_namespaces:
201     \__tag_tree_write_structelements: %this is rather slow!!
202     \__tag_tree_write_structtreeroot:
203 }

```

(End definition for \\_\_tag\_finish\_structure:.)

## 1.7 StructParents entry for Page

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

```

204 \hook_gput_code:nnn{begindocument}{tagpdf}
205 {
206     \bool_if:NT\g__tag_active_tree_bool
207     {
208         \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
209         {
210             \pdfmanagement_add:nnx
211             { Page }
212             {StructParents}
213             {\int_eval:n { \g_shipout_readonly_int}}
214         }
215     }
216 }
217 </tree>

```

# 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.

A		H	
\AddToHook	11, 14	hook commands:	
		\hook_gput_code:nnn	5, 19, 42, 204, 208
B		I	
bool commands:		int commands:	
\bool_if:NTF	7, 21, 118, 206	\int_compare:nTF	65
		\int_eval:n	76, 213
		\int_gset:Nn	44
		\int_step_inline:nnnn	35, 59, 62, 79
		\int_use:N	111
C		iow commands:	
c@g internal commands:		\iow_newline:	156
\c@g__tag_parenttree_obj_int	41		
\c@g__tag_struct_abs_int	35		
cs commands:		M	
\cs_new_protected:Npn	29,	msg commands:	
	33, 49, 57, 103, 116, 132, 139, 172, 195	\msg_warning:nn	93
D		N	
\directlua	107	\newcounter	41

P	
pdf commands:	
\pdf_object_new:nn	18, 40, 131, 161, 171
\pdf_object_ref:n	26, 168, 183
\pdf_object_write:nn	126, 134, 162, 178, 185, 190
pdfdict commands:	
\pdfdict_gput:nnn	182
\pdfdict_if_empty:nTF	176
\pdfdict_use:n	136, 180, 187
pdfmanagement commands:	
\pdfmanagement_add:nnn	23, 210
prop commands:	
\prop_clear:N	61
\prop_get:NnNTF	84
\prop_item:Nn	71, 88, 147
\prop_map_function:NN	82
\prop_map_inline:Nn	174
\prop_map_tokens:Nn	192
\prop_put:Nnn	68
\ProvidesExplPackage	3
S	
seq commands:	
\seq_gremove_duplicates:N	142
\seq_set_map:NNn	143
\seq_use:Nn	154
shipout commands:	
\g_shipout_readonly_int	111, 213
sys commands:	
\sys_if_output_pdf:TF	9
T	
tag internal commands:	
\g__tag_active_struct_bool	21
\g__tag_active_tree_bool	7, 206
\g__tag_attr_class_content_tl	141, 152, 159, 164
\g__tag_attr_class_used_seq	142, 143
\g__tag_attr_entries_prop	148
\__tag_finish_structure:	11, 14, 195, 195
\g__tag_mc_parenttree_prop	71
\g__tag_mode_lua_bool	118
\__tag_parenttree_add_objr:nn	49, 49
\l__tag_parenttree_content_tl	56, 74, 86, 97, 105, 125, 128
\g__tag_parenttree_objr_tl	48, 51, 125
\__tag_prop_count:nn	82
\__tag_prop_gput:Nnn	165
\__tag_ref_value:nnn	66, 70
\__tag_ref_value_lastpage:nn	46, 59, 62
\g__tag_role_NS_prop	174, 192
\__tag_struct_write_obj:n	31, 37
\l__tag_tmpa_prop	61, 69, 82, 84
\l__tag_tmpa_seq	143, 155
\l__tag_tmpa_tl	84, 85, 88
\__tag_tree_fill_parenttree:	57, 57, 123
\__tag_tree_lua_fill_parenttree:	103, 103, 120
\__tag_tree_write_classmap:	139, 139, 199
\__tag_tree_write_namespaces:	172, 172, 200
\__tag_tree_write_parenttree:	116, 116, 197
\__tag_tree_write_rolemap:	132, 132, 198
\__tag_tree_write_structelements:	33, 33, 201
\__tag_tree_write_structtreeroot:	29, 29, 202
tag/struct/0 internal commands:	
__tag/struct/0	18
tag/tree/namespaces internal commands:	
__tag/tree/namespaces	171
tag/tree/parenttree internal commands:	
__tag/tree/parenttree	40
tag/tree/rolemap internal commands:	
__tag/tree/rolemap	131
tl commands:	
\c_space_tl	53, 76, 77, 89, 128, 145
\tl_gclear:N	141
\tl_gput_right:Nn	51
\tl_gset:Nn	152
\tl_if_empty:NnTF	158
\tl_new:N	48, 56
\tl_put_right:Nn	74, 86, 96, 125
\tl_set:Nn	105
U	
use commands:	
\use_ii:nn	192