

tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer[†]

Released 2021-06-14

Contents

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

*This file describes v0.82, last revised 2021-06-14.

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

| | | |
|---|--|-----------|
| 3 | Messages | 12 |
| 3.1 | Messages related to mc-chunks | 12 |
| 3.2 | Messages related to mc-chunks | 13 |
| 3.3 | Attributes | 14 |
| 3.4 | Roles | 14 |
| 3.5 | Miscellaneous | 14 |
| 4 | Retrieving data | 15 |
| 5 | User conditionals | 15 |
| 6 | Internal checks | 15 |
| 6.1 | checks for active tagging | 15 |
| 6.2 | Checks related to stuctures | 16 |
| 6.3 | Checks related to roles | 17 |
| 6.4 | Check related to mc-chunks | 17 |
| 6.5 | Miscellaneous | 20 |
| | | |
| II The tagpdf-user module | | |
| Code related to L ^A T _E X2e user commands and document com- | | |
| mands | | |
| part of the tagpdf package | | 21 |
| 1 | Setup commands | 21 |
| 2 | Commands related to mc-chunks | 21 |
| 3 | Commands related to structures | 21 |
| 4 | Debugging | 22 |
| 5 | Extension commands | 22 |
| 5.1 | Fake space | 22 |
| 5.2 | Paratagging | 22 |
| 5.3 | Link tagging | 23 |
| 6 | User commands and extensions of document commands | 23 |
| 7 | Setup and preamble commands | 23 |
| 8 | Commands for the mc-chunks | 23 |
| 9 | Commands for the structure | 24 |
| 10 | Debugging | 25 |
| 11 | Commands to extend document commands | 27 |
| 11.1 | Fake space | 27 |
| 11.2 | Paratagging | 27 |
| 11.3 | Links | 28 |

| | | |
|------------|--|-----------|
| III | The <code>tagpdf-tree</code> module | |
| | Commands trees and main dictionaries | |
| | part of the <code>tagpdf</code> package | 30 |
| 1 | Trees, pdfmanagement and finalization code | 30 |
| 1.1 | Catalog: MarkInfo and StructTreeRoot | 30 |
| 1.2 | Writing structure elements | 31 |
| 1.3 | ParentTree | 31 |
| 1.4 | Rolemap dictionary | 34 |
| 1.5 | Classmap dictionary | 34 |
| 1.6 | Namespaces | 35 |
| 1.7 | Finishing the structure | 36 |
| 1.8 | StructParents entry for Page | 36 |
| IV | The <code>tagpdf-mc</code> module | |
| | Code related to Marked Content (<code>mc-chunks</code>) | |
| | part of the <code>tagpdf</code> package | 37 |
| 1 | Public Commands | 37 |
| 2 | Public keys | 38 |
| 3 | Marked content code – shared | 38 |
| 3.1 | Variables and counters | 39 |
| 3.2 | Functions | 40 |
| 3.3 | Keys | 41 |
| 4 | Marked content code – generic mode | 42 |
| 4.1 | Variables | 42 |
| 4.2 | Functions | 43 |
| 4.3 | Keys | 47 |
| 5 | Marked content code – luamode code | 48 |
| 5.1 | Commands | 49 |
| 5.2 | Key definitions | 53 |
| V | The <code>tagpdf-struct</code> module | |
| | Commands to create the structure | |
| | part of the <code>tagpdf</code> package | 56 |
| 1 | Public Commands | 56 |
| 2 | Public keys | 56 |
| 2.1 | Keys for the structure commands | 56 |
| 2.2 | Setup keys | 58 |
| 3 | Variables | 58 |
| 3.1 | Variables used by the keys | 60 |

| | | |
|-------------|---|------------|
| 4 | Commands | 60 |
| 4.1 | Initialization of the StructTreeRoot | 61 |
| 4.2 | Handlings kids | 62 |
| 5 | Keys | 66 |
| 6 | User commands | 70 |
| 7 | Attributes and attribute classes | 73 |
| 7.1 | Variables | 73 |
| 7.2 | Commands and keys | 73 |
| | | |
| VI | The <code>tagpdf-luatex.def</code> Driver for luatex part of the tagpdf package | 76 |
| 1 | Loading the lua | 76 |
| | | |
| VII | The <code>tagpdf-roles</code> module Tags, roles and namespace code part of the tagpdf package | 91 |
| 1 | Code related to roles and structure names | 91 |
| 1.1 | Variables | 91 |
| 1.2 | Namespaces | 92 |
| 1.3 | Data | 93 |
| 1.4 | Adding new tags and rolemapping | 99 |
| 1.4.1 | pdf 1.7 and earlier | 99 |
| 1.4.2 | The pdf 2.0 version | 100 |
| 1.5 | Key-val user interface | 101 |
| | | |
| VIII | The <code>tagpdf-space</code> module code related to real space chars part of the tagpdf package | 103 |
| 1 | Code for interword spaces | 103 |
| | | |
| | Index | 105 |

1 Initialization and test if pdfmanagement is active.

```
1 <@@=tag>
2 <*package>
3 \ProvidesExplPackage {tagpdf} {2021-06-14} {0.82}
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`
`\l__tag_tmpa_str`
`\l__tag_tmpa_prop`
`\l__tag_tmpa_seq`
`\l__tag_tmpb_seq`
`\l__tag_tmpa_clist`
`\l__tag_tmpa_int`

A few temporary variables

```

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    {tagabspace,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_mc_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 mc-boolean activates `\tag_mc_begin:n`, the `\g__tag_active_tree_bool` tree-boolean activates writing the finish code and the pdfmanagement related commands, the `\g__tag_active_struct_bool` 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_mc_bool
81 \bool_new:N \g__tag_active_tree_bool
82 \bool_new:N \g__tag_active_struct_bool

```

(End definition for `\g__tag_active_mc_bool`, `\g__tag_active_tree_bool`, and `\g__tag_active_struct_bool`.)

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

```

83 \bool_new:N \l__tag_active_mc_bool
84 \bool_set_true:N \l__tag_active_mc_bool
85 \bool_new:N \l__tag_active_struct_bool
86 \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.

```

87 \bool_new:N \g__tag_tagunmarked_bool

```

(End definition for `\g__tag_tagunmarked_bool`.)

6 Variants of l3 commands

```

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

```

97 \ref_attribute_gset:nnnn { tagstruct } {0} { now }
98 { \int_use:N \c@g__tag_struct_abs_int }
99 \ref_attribute_gset:nnnn { tagstructobj } {} { now }
100 {
101   \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
102   {
103     \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
104   }
105 }
106 \ref_attribute_gset:nnnn { tagabspage } {0} { shipout }
107 { \int_use:N \g_shipout_readonly_int }
108 \ref_attribute_gset:nnnn { tagmcabs } {0} { now }
109 { \int_use:N \c@g__tag_MCID_abs_int }
110 \ref_attribute_gset:nnnn {tagmcid } {0} { now }
111 { \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.
112 \cs_new_protected:Npn \__tag_ref_label:nn #1 #2 %#1 label, #2 name of list mc or struct
113 {
114   \@bsphack
115   \ref_label:nv {#1}{c__tag_ref#2_clist}
116   \@esphack
117 }
118 \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.

```

119 \cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 %#1 label, #2 attribute, #3 default
120 {
121   \ref_value:nnn {#1}{#2}{#3}
122 }
123 \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.

```

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

```

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

```

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

```

```

148     }
149     \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-mc  Keys to (globally) activate tagging.
activate-tree 150 \keys_define:nn { __tag / setup }
activate-struct 151 {
activate-all 152     activate-mc      .bool_gset:N = \g__tag_active_mc_bool,
activate      153     activate-tree   .bool_gset:N = \g__tag_active_tree_bool,
              154     activate-struct .bool_gset:N = \g__tag_active_struct_bool,
              155     activate-all   .meta:n = {activate-mc,activate-tree,activate-struct},
              156     activate      .meta:n = {activate-mc,activate-tree,activate-struct},

```

(End definition for activate-mc 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.

```

157     log          .choice:,
158     log / none   .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
159     log / v      .code:n = {\int_set:Nn \l__tag_loglevel_int { 1 }},
160     log / vv     .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
161     log / vvv    .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
162     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.

```

163     tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
164     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.

```

165     tabsorder .choice:,
166     tabsorder / row .code:n =
167         \pdfmanagement_add:nnn { Page } {Tabs}{/R},
168     tabsorder / column .code:n =
169         \pdfmanagement_add:nnn { Page } {Tabs}{/C},
170     tabsorder / structure .code:n =
171         \pdfmanagement_add:nnn { Page } {Tabs}{/S},
172     tabsorder / none .code:n =
173         \pdfmanagement_remove:nn {Page} {Tabs},
174     tabsorder .initial:n = structure,
175     uncompress .code:n = { \pdf_uncompress: },
176 }

```

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

12 loading of engine/more dependent code

```
177 \sys_if_engine luatex:T
178 {
179   \file_input:n {tagpdf-luatex.def}
180 }
181 \</package>
182 \<*/mcloading>
183 \bool_if:NTF \g__tag_mode_lua_bool
184 {
185   \RequirePackage {tagpdf-mc-code-lua}
186 }
187 {
188   \RequirePackage {tagpdf-mc-code-generic} %
189 }
190 \</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 log-levels

| command/message | log-level | type | note |
|--|-----------|----------|----------|
| <code>\showtagpdfmcdata</code> | 0 | log/term | lua-only |
| 1 <code><@@=tag></code> | | | |
| 2 <code><*header></code> | | | |
| 3 <code>\ProvidesExplPackage {tagpdf-checks-code} {2021-06-14} {0.82}</code> | | | |
| 4 <code>{part of tagpdf - code related to checks, conditionals, debugging and messages}</code> | | | |
| 5 <code></header></code> | | | |

3 Messages

3.1 Messages related to mc-chunks

`mc-nested` This message is issued when 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)

9 `\msg_new:nnn { tag } {mc-label-unknown} { label~#1-unknown~~rerun }`

(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.

```
10 \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.

```
11 \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.

```
12 \msg_new:nnn { tag } {mc-pushed} { #1~has~been~pushed~to~the~mc~stack}
13 \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.

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

(End definition for mc-current. This function is documented on page 22.)

3.2 Messages related to mc-chunks

struct-no-objnum Should not happen ...

```
20 \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.

```
21 \msg_new:nnn { tag }
22 {struct-faulty-nesting}
23 { 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.

```
24 \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

```
25 \msg_new:nnn { tag } {struct-used-twice}
26 { 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.

```
27 \msg_new:nnn { tag } {struct-label-unknown}  
28 { 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

```
29 \msg_new:nnn { tag } {struct-show-closing}  
30 { 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

```
31 \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 32 \msg_new:nnn { tag } {role-missing} { tag~#1~has~no~role~assigned }  
role-unknown-tag 33 \msg_new:nnn { tag } {role-unknown} { role~#1~is~not~known }  
34 \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 35 \msg_new:nnn { tag } {role-tag} { mapping~tag~#1~to~role~#2 }  
36 \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.

```
37 \msg_new:nnn { tag } {tree-mcid-index-wrong}  
38 { something~is~wrong~with~the~mcid~--rerun }
```

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

obj-write-num An info message, useful for reporting.

```
39 \msg_new:nnn { tag } {obj-write-num} { write~obj~#1~to~pdf }
```

(End definition for obj-write-num. This function is documented on page ??.)

sys-no-interwordspace Currently only pdf_lat_{ex} and lua_lat_{ex} have some support for real spaces.

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

```
42 \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 12.)

5 User conditionals

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

```
43 \prg_new_conditional:Npnn \tag_if_active: { p , T , TF , F }
44 {
45     \bool_lazy_all:nTF
46     {
47         {\g__tag_active_struct_bool}
48         {\g__tag_active_mc_bool}
49         {\g__tag_active_tree_bool}
50         {\l__tag_active_struct_bool}
51         {\l__tag_active_mc_bool}
52     }
53     {
54         \prg_return_true:
55     }
56     {
57         \prg_return_false:
58     }
59 }
```

(End definition for \tag_if_active:TF. This function is documented on page 12.)

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.

```
60 \prg_new_conditional:Npnn \__tag_check_if_active_mc: { T,F,TF }
61 {
62     \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
63     {
64         \prg_return_true:
65     }
66     {
67         \prg_return_false:
68     }
69 }
70 \prg_new_conditional:Npnn \__tag_check_if_active_struct: { T,F,TF }
```

```

71 {
72   \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
73   {
74     \prg_return_true:
75   }
76   {
77     \prg_return_false:
78   }
79 }

```

(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.

```

80 \cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
81 {
82   \prop_if_in:cnF { g__tag_struct_#1_prop }
83   {S}
84   {
85     \msg_error:nn { tag } {struct-missing-tag}
86   }
87 }

```

(End definition for __tag_check_structure_has_tag:n.)

__tag_check_structure_tag:N This checks if the name of the tag is known.

```

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

```

(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.

```

95 \cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
96 {
97   \msg_info:nnn { tag } {struct-show-closing} {#1}
98 }
99
100 \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.

```

101 \cs_new_protected:Npn \__tag_check_no_open_struct:
102 {
103   \msg_error:nn { tag } {struct-faulty-nesting}
104 }

```


(End definition for `_tag_check_no_open_struct:`.)

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

```

105 \cs_new_protected:Npn \_tag\_check\_struct\_used:n #1 %#1 label
106 {
107   \prop_get:cnNT
108     {g\_tag\_struct\_\_tag\_ref\_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}\_prop}
109     {P}
110     \l\_tmpa\_tl
111     {
112       \msg_warning:nnn { tag } {struct-used-twice} {#1}
113     }
114 }
```

(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.

```

115 \cs_new_protected:Npn \_tag\_check\_add\_tag\_role:nn #1 #2 %#1 tag, #2 role
116 {
117   \tl_if_empty:nTF {#2}
118   {
119     \msg_warning:nnn { tag } {role-missing} {#1}
120   }
121   {
122     \prop_get:NnNTF \g\_tag\_role\_tags\_prop {#2} \l\_tmpa\_tl
123     {
124       \msg_info:nnnn { tag } {role-tag} {#1} {#2}
125     }
126     {
127       \msg_warning:nnn { tag } {role-unknown} {#2}
128     }
129   }
130 }
```

(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.

```

\_tag\_check\_mc\_if\_open:
131 \cs_new_protected:Npn \_tag\_check\_mc\_if\_nested:
132 {
133   \_tag\_mc\_if\_in:T
134   {
135     \msg_warning:nnx { tag } {mc-nested} { \_tag\_get\_mc\_abs\_cnt: }
136   }
137 }
138
139 \cs_new_protected:Npn \_tag\_check\_mc\_if\_open:
140 {
141   \_tag\_mc\_if\_in:F
142   {
143     \msg_warning:nnx { tag } {mc-not-open} { \_tag\_get\_mc\_abs\_cnt: }
```

```

144     }
145 }

```

(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.

```

146 \cs_new_protected:Npn \_tag\_check\_mc\_pushed\_popped:nn #1 #2
147 {
148   \int_compare:nNtT
149     { \l__tag_loglevel_int } = { 2 }
150     { \msg_info:nnx {tag}{mc-#1}{#2} }
151   \int_compare:nNtT
152     { \l__tag_loglevel_int } > { 2 }
153     {
154       \msg_warning:nnx {tag}{mc-#1}{#2}
155       \seq_log:N \g__tag_mc_stack_seq
156     }
157 }

```

(End definition for `_tag_check_mc_pushed_popped:nn.`)

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

```

158 \cs_new_protected:Npn \_tag\_check\_mc\_tag:N #1   %#1 is var with a tag name in it
159 {
160   \tl_if_empty:NT #1
161   {
162     \msg_error:nnx { tag } {mc-tag-missing} { \_tag\_get\_mc\_abs\_cnt: }
163   }
164   \prop_if_in:Nof \g__tag_role_tags_NS_prop {#1}
165   {
166     \msg_warning:nnx { tag } {role-unknown-tag} {#1}
167   }
168 }

```

(End definition for `_tag_check_mc_tag:N.`)

`\g__tag_check_mc_used_seq` This variable holds the list of used mc. It will hopefully be rather short, so checking the seq is not to slow.

```

169 \seq_new:N \g__tag\_check\_mc\_used\_seq

```

(End definition for `\g__tag_check_mc_used_seq.`)

`_tag_check_mc_used:n` This checks if a mc is used twice.

```

170 \cs_new_protected:Npn \_tag\_check\_mc\_used:n #1
171 {
172   \seq_if_in:NnTF \g__tag\_check\_mc\_used\_seq {#1}
173   {
174     \msg_warning:nnn { tag } {mc-used-twice} {#1}
175   }
176   {
177     \seq_gput_right:Nx \g__tag\_check\_mc\_used\_seq {#1}
178   }
179 }

```

(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.

```

180 \cs_new_protected:Npn \_tag\_check\_show\_MCID\_by\_page:
181 {
182   \tl_set:Nx \l__tag\_tmpa\_tl
183   {
184     \_tag\_ref\_value\_lastpage:nn
185     {abspage}
186     {-1}
187   }
188   \int_step_inline:nnnn {1}{1}
189   {
190     \l__tag\_tmpa\_tl
191   }
192   {
193     \seq_clear:N \l_tmpa\_seq
194     \int_step_inline:nnnn
195     {1}
196     {1}
197     {
198       \_tag\_ref\_value\_lastpage:nn
199       {tagmcabs}
200       {-1}
201     }
202     {
203       \int_compare:nT
204       {
205         \_tag\_ref\_value:enn
206         {mcid-###1}
207         {tagabspage}
208         {-1}
209         =
210         ##1
211       }
212       {
213         \seq_gput_right:Nx \l_tmpa\_seq
214         {
215           Page##1-###1-
216           \_tag\_ref\_value:enn
217           {mcid-###1}
218           {tagmcid}
219           {-1}
220         }
221       }
222     }
223     \seq_show:N \l_tmpa\_seq
224   }
225 }

```

(End definition for _tag_check_show_MCID_by_page:.)

6.5 Miscellaneous

`_tag_check_record_pdfobj_num:n` This writes the object numbers. Not sure if needed, currently unused.

```
226 \cs_new_protected:Npn \_tag_check_record_pdfobj_num:n #1
227 {
228   \int_compare:nT { \l__tag_loglevel_int >= 3 }
229   {
230     \msg_info:nx { tag } {obj-write-num} {#1}
231   }
232 }
233 </package>
```

(End definition for _tag_check_record_pdfobj_num:n.)

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} {2021-06-14} {0.82}
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 21.)

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 21.)

`\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 21.)

`\tagpdfifluatexTF` I should deprecate them ...

```

\tagpdfifluatexT 46 \cs_set_eq:NN\tagpdfifluatexTF \sys_if_engine luatex:TF
\tagpdfifpdfltexTF 47 \cs_set_eq:NN\tagpdfifluatexT \sys_if_engine luatex:T
48 \cs_set_eq:NN\tagpdfifpdfltexT \sys_if_engine pdfltex: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 22.)

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 22.)

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 22.)

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 22.)

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 Fake space

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

```

135 \sys_if_engine_luatex:T
136 {
137   \NewDocumentCommand\pdffakespace { }
138   {
139     \__tag_fakespace:
140   }
141 }
```

(End definition for `\pdffakespace`. This function is documented on page 22.)

11.2 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
\g__tag_para_int
142 \bool_new:N \l__tag_para_bool
143 \bool_new:N \l__tag_para_show_bool
144 \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 typesetting if `paratagging-show` is used. The small numbers are boxes and they have a (small) height.

```

145 \keys_define:nn { __tag / setup }
146 {
147   paratagging .bool_set:N = \l__tag_para_bool,
148   paratagging-show .bool_set:N = \l__tag_para_show_bool,
149 }
150
```

(End definition for `paratagging` and `paratagging-show`. These functions are documented on page 23.)

This fills the para hooks with the needed code.

```

151 \AddToHook{para/begin}
152 {
153   \int_gincr:N \g__tag_para_int
154   \bool_if:NT \l__tag_para_bool
155   {
```

```

156     \tag_struct_begin:n {tag=P}
157     \bool_if:NT \l__tag_para_show_bool
158     { \tag_mc_begin:n{artifact}
159       \llap{\color_select:n{red}\tiny\int_use:N\g__tag_para_int\ }
160       \tag_mc_end:
161     }
162     \tag_mc_begin:n {tag=P}
163   }
164 }
165 \AddToHook{para/end}
166 {
167   \bool_if:NT \l__tag_para_bool
168   {
169     \tag_mc_end:
170     \bool_if:NT \l__tag_para_show_bool
171     { \tag_mc_begin:n{artifact}
172       \rlap{\color_select:n{red}\tiny\int_use:N\g__tag_para_int}
173       \tag_mc_end:
174     }
175     \tag_struct_end:
176   }
177 }

```

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

```

178 \newcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
179 \newcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}

```

(End definition for `\tagpdfparaOn` and `\tagpdfparaOff`. These functions are documented on page 23.)

11.3 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.

```

180 \hook_gput_code:nnn
181 {pdfannot/link/URI/before}
182 {tagpdf}
183 {
184   \tag_mc_end_push:
185   \tag_struct_begin:n { tag=Link }
186   \tag_mc_begin:n { tag=Link }
187   \pdfannot_dict_put:nnx
188   { link/URI }
189   { StructParent }
190   { \tag_struct_parent_int: }
191 }
192
193 \hook_gput_code:nnn
194 {pdfannot/link/URI/after}
195 {tagpdf}
196 {
197   \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
198   \tag_mc_end:

```

```

199     \tag_struct_end:
200     \tag_mc_begin_pop:n{ }
201 }
202
203 \hook_gput_code:nnn
204 {pdfannot/link/GoTo/before}
205 {tagpdf}
206 {
207     \tag_mc_end_push:
208     \tag_struct_begin:n{tag=Link}
209     \tag_mc_begin:n{tag=Link}
210     \pdfannot_dict_put:nnx
211     { link/GoTo }
212     { StructParent }
213     { \tag_struct_parent_int: }
214 }
215
216 \hook_gput_code:nnn
217 {pdfannot/link/GoTo/after}
218 {tagpdf}
219 {
220     \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
221     \tag_mc_end:
222     \tag_struct_end:
223     \tag_mc_begin_pop:n{ }
224 }
225 }
226
227 % "alternative descriptions " for PAX3. How to get better text here??
228 \pdfannot_dict_put:nnn
229 { link/URI }
230 { Contents }
231 { (url) }
232
233 \pdfannot_dict_put:nnn
234 { link/GoTo }
235 { Contents }
236 { (ref) }
237
</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} {2021-06-14} {0.82}
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/Namespace_##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/Namespace_##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         \_tag_tree_write_parenttree:
215         \_tag_tree_write_rolemap:
216         \_tag_tree_write_classmap:
217         \_tag_tree_write_namespaces:
218         \_tag_tree_write_structelements: %this is rather slow!!
219         \_tag_tree_write_structtreeroot:
220     }
221 }

```

(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.

```

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

```

Part IV

The **tagpdf-mc** module

Code related to Marked Content (mc-chunks)

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} {2021-06-14} {0.82}
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_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

```
11 \__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:

```
12 \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.

```
13 \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`

```
14 \bool_new:N \l__tag_mc_key_stash_bool
```

```
15 \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 `pdflatex` now?

`\g__tag_mc_key_tag_tl`

`\l__tag_mc_key_label_tl`

```
16 \tl_new:N \l__tag_mc_key_tag_tl
```

```
17 \tl_new:N \g__tag_mc_key_tag_tl
```

```
18 \tl_new:N \l__tag_mc_key_label_tl
```

```
19 \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

`tagabspage`: 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`.

```
20 \cs_new:Nn \_tag_mc_handle_mc_label:n
21 {
22   \_tag_ref_label:en{tagpdf-#1}{mc}
23 }
```

(End definition for `_tag_mc_handle_mc_label:n`.)

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

`\tag_mc_artifact_group_end:`

```
24 \cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1
25 {
26   \tag_mc_end_push:
27   \tag_mc_begin:n {artifact=#1}
28   \tag_stop_group_begin:
29 }
30
31 \cs_new_protected:Npn \tag_mc_artifact_group_end:
32 {
33   \tag_stop_group_end:
34   \tag_mc_end:
35   \tag_mc_begin_pop:n{}
36 }
```

(End definition for `\tag_mc_artifact_group_begin:n` and `\tag_mc_artifact_group_end:`. These functions are documented on page 37.)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

```
37 \cs_new_protected:Npn \tag_mc_end_push:
38 {
39   \_tag_check_if_active_mc:T
40   {
41     \_tag_mc_if_in:TF
42     {
43       \seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
44       \_tag_check_mc_pushed_popped:nn
45         { pushed }
46         { \tag_get:n {mc_tag} }
47       \tag_mc_end:
48     }
49     {
50       \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
51       \_tag_check_mc_pushed_popped:nn { pushed }{-1}
52     }
53   }
```



```

54 }
55
56 \cs_new_protected:Npn \tag_mc_begin_pop:n #1
57 {
58   \__tag_check_if_active_mc:T
59   {
60     \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
61     {
62       \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
63       {
64         \__tag_check_mc_pushed_popped:nn {popped}{-1}
65       }
66       {
67         \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
68         \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
69       }
70     }
71     {
72       \__tag_check_mc_pushed_popped:nn {popped}{empty-stack,~nothing}
73     }
74   }
75 }

```

(End definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 37.)

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 76 \keys_define:nn { __tag / mc }
__artifact-type 77 {
78   stash .bool_set:N = \l__tag_mc_key_stash_bool,
79   __artifact-bool .bool_set:N = \l__tag_mc_artifact_bool,
80   __artifact-type .choice:,
81   __artifact-type / pagination .code:n =
82   {
83     \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
84   },
85   __artifact-type / layout .code:n =
86   {
87     \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
88   },
89   __artifact-type / page .code:n =
90   {
91     \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
92   },
93   __artifact-type / background .code:n =
94   {
95     \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
96   },
97   __artifact-type / notype .code:n =
98   {

```

```

99      \tl_set:Nn \l__tag_mc_artifact_type_tl {}
100    },
101    __artifact-type /      .code:n      =
102    {
103      \tl_set:Nn \l__tag_mc_artifact_type_tl {}
104    },
105  }

```

(End definition for `stash`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 57.)

```

106 </shared>

```

4 Marked content code – generic mode

```

107 <*generic>
108 \ProvidesExplPackage {tagpdf-mc-code-generic} {2021-06-14} {0.82}
109 {part of tagpdf - code related to marking chunks - generic mode}
110 </generic>

```

4.1 Variables

`\g__tag_in_mc_bool` This boolean records if a mc is open, to test nesting.

```

111 <*generic>
112 \bool_new:N \g__tag_in_mc_bool

(End definition for \g__tag_in_mc_bool.)

```

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

```

113 \__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.

```

114 \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

```

115 \tl_new:N \l__tag_mc_tmpa_tl

(End definition for \l__tag_mc_tmpa_tl.)

```

4.2 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_p:TF
\tag_mc_if_in_p:TF
\tag_mc_if_in_p:TF
116 \prg_new_conditional:Nnn \_tag_mc_if_in: {p,T,F,TF}
117 {
118   \bool_if:NTF \g__tag_in_mc_bool
119   { \prg_return_true: }
120   { \prg_return_false: }
121 }
122
123 \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 37.)

```

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

```

124 % #1 tag, #2 properties
125 \cs_set_eq:NN \_tag_mc_bmc:n \pdf_bmc:n
126 \cs_set_eq:NN \_tag_mc_emc: \pdf_emc:
127 \cs_set_eq:NN \_tag_mc_bdc:nn \pdf_bdc:nn
128 \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 number value for the MCID: they must be numbered by page starting with 0 and then successively. The first argument is the tag, e.g. P or Span, the second is used to pass more properties. We also define a wrapper around the low-level command as luamode will need something different.
`_tag_mc_bdc_mcid:n`
`_tag_mc_handle_mcid:nn`
`_tag_mc_handle_mcid:VV`

```

129 \cs_new_protected:Npn \_tag_mc_bdc_mcid:nn #1 #2
130 {
131   \int_gincr:N \c@g__tag_MCID_abs_int
132   \tl_set:Nx \l__tag_mc_ref_abspage_tl
133   {
134     \_tag_ref_value:enn %3 args
135     {
136       mcid-\int_use:N \c@g__tag_MCID_abs_int
137     }
138     { tagabspage }
139     {-1}
140   }
141   \prop_get:NoNTF
142   \g__tag_MCID_byabspage_prop
143   {
144     \l__tag_mc_ref_abspage_tl
145   }
146   \l__tag_mc_tmpa_tl
147   {
148     %key already present, use value for MCID and add 1 for the next
149     \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }

```

```

150     \__tag_prop_gput:Nxx
151     \g__tag_MCID_byabspage_prop
152     { \l__tag_mc_ref_abspage_tl }
153     { \int_eval:n {\l__tag_mc_tmpa_tl +1} }
154   }
155   {
156     %key not present, set MCID to 0 and insert 1
157     \int_gzero:N \g__tag_MCID_tmp_bypage_int
158     \__tag_prop_gput:Nxx
159     \g__tag_MCID_byabspage_prop
160     { \l__tag_mc_ref_abspage_tl }
161     {1}
162   }
163   \__tag_ref_label:en
164   {
165     mcid-\int_use:N \c@g__tag_MCID_abs_int
166   }
167   { mc }
168   \__tag_mc_bdc:nx
169   {#1}
170   { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
171 }
172 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
173 {
174   \__tag_mc_bdc_mcid:nn {#1} {}
175 }
176
177 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %1 tag, #2 properties
178 {
179   \__tag_mc_bdc_mcid:nn {#1} {#2}
180 }
181
182 \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 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,

```

183 \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
184 {
185   \__tag_check_mc_used:n {#1}
186   \__tag_struct_kid_mc_gput_right:nn
187   { \g__tag_struct_stack_current_tl }
188   {#1}
189   \prop_gput:Nxx \g__tag_mc_parenttree_prop
190   {#1}
191   { \g__tag_struct_stack_current_tl }
192 }

```

(End definition for __tag_mc_handle_stash:n.)

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

__tag_mc_bmc_artifact:n

__tag_mc_handle_artifact:N

```

193 \cs_new_protected:Npn \__tag_mc_bmc_artifact:
194 {
195   \__tag_mc_bmc:n {Artifact}
196 }
197 \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
198 {
199   \__tag_mc_bdc:nn {Artifact}{/Type/#1}
200 }
201 \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
202   % #1 is a var containing the artifact type
203 {
204   \tl_if_empty:NTF #1
205     { \__tag_mc_bmc_artifact: }
206     { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
207 }

```

(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.

```

208 \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:

```

209 \cs_new_protected:Npn \tag_mc_begin:n #1 %#1 keyval
210 {
211   \__tag_check_if_active_mc:T
212   {
213     \group_begin: %hm
214     \__tag_check_mc_if_nested:
215     \bool_gset_true:N \g__tag_in_mc_bool
216     \keys_set:nn { __tag / mc } {#1}
217     \bool_if:NTF \l__tag_mc_artifact_bool
218       { %handle artifact
219         \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
220       }
221     { %handle mcid type
222       \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
223       \__tag_mc_handle_mcid:VV
224         \l__tag_mc_key_tag_tl
225         \l__tag_mc_key_properties_tl
226       \tl_if_empty:NF {\l__tag_mc_key_label_tl}
227         {
228           \exp_args:NV
229           \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
230         }
231       \bool_if:NF \l__tag_mc_key_stash_bool
232         {
233           \__tag_mc_handle_stash:n { \int_use:N \c@g__tag_MCID_abs_int }
234         }

```

```

235     }
236     \group_end:
237 }
238 }
239 \cs_new_protected:Nn \tag_mc_end:
240 {
241     \__tag_check_if_active_mc:T
242     {
243         \__tag_check_mc_if_open:
244         \bool_gset_false:N \g__tag_in_mc_bool
245         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
246         \__tag_mc_emc:
247     }
248 }

```

(End definition for `\tag_mc_begin:n` and `\tag_mc_end:.` These functions are documented on page 37.)

`\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 the claim about the warning right??? TODO: is testing for struct the right test?

```

249 \cs_new_protected:Npn \tag_mc_use:n #1 %#1: label name
250 {
251     \__tag_check_if_active_struct:T
252     {
253         \tl_set:Nx \l__tag_tmpa_tl { \__tag_ref_value:enn{tagpdf-#1}{tagmcabs}{ } }
254         \tl_if_empty:NTF\l__tag_tmpa_tl
255         {
256             \msg_warning:nnn {tag} {mc-label-unknown} {#1}
257         }
258         {
259             \prop_gput:Nxx
260             \g__tag_mc_parenttree_prop
261             {
262                 \l__tag_tmpa_tl
263             }
264             {
265                 \g__tag_struct_stack_current_tl
266             }
267             \__tag_struct_kid_mc_gput_right:nn
268             {
269                 \g__tag_struct_stack_current_tl
270             }
271             {
272                 \l__tag_tmpa_tl
273             }
274         }
275     }
276 }

```

(End definition for `\tag_mc_use:n`. This function is documented on page 37.)

4.3 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
277 \keys_define:nn { __tag / mc }
278 {
279   tag .code:n = % the name (H,P,Span) etc
280   {
281     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
282     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
283   },
284   raw .code:n =
285   {
286     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
287   },
288   alttext .code:n = % Alt property
289   {
290     \str_set_convert:Nnon
291     \l__tag_tmpa_str
292     { #1 }
293     { default }
294     { utf16/hex }
295     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
296     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
297   },
298   alttext-o .code:n = % Alt property
299   {
300     \str_set_convert:Noon
301     \l__tag_tmpa_str
302     { #1 }
303     { default }
304     { utf16/hex }
305     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
306     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
307   },
308   actualtext .code:n = % ActualText property
309   {
310     \str_set_convert:Nnon
311     \l__tag_tmpa_str
312     { #1 }
313     { default }
314     { utf16/hex }
315     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
316     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
317   },
318   actualtext-o .code:n = % ActualText property
319   {
320     \str_set_convert:Noon
321     \l__tag_tmpa_str
322     { #1 }
323     { default }
324     { utf16/hex }

```

```

325     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
326     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
327   },
328   label .tl_set:N      = \l__tag_mc_key_label_tl,
329   artifact .code:n     =
330   {
331     \exp_args:Nnx
332     \keys_set:nn
333     { __tag / mc }
334     { __artifact-bool, __artifact-type=#1 }
335   },
336   artifact .default:n  = {notype}
337 }
338 </generic>

```

(End definition for tag and others. These functions are documented on page 56.)

5 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.

```

339 <*luamode>
340 \ProvidesExplPackage {tagpdf-mc-code-lua} {2021-06-14} {0.82}
341 {tagpdf - mc code only for the luamode }
342 </luamode>

```

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

```

343 <*luamode>
344 \hook_gput_code:nnn{begindocument}{tagpdf/mc}

```



```

345 {
346   \bool_if:NT\g__tag_active_mc_bool
347   {
348     \lua_now:e
349     {
350       if~luatexbase.callbacktypes.pre_shipout_filter~then~
351         luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
352           ltx.__tag.func.mark_shipout(TAGBOX)~return~true~
353         end, "tagpdf")~
354       end
355     }
356   \lua_now:e
357   {
358     if~luatexbase.callbacktypes.pre_shipout_filter~then~
359       token.get_next()~
360     end
361   }~\@secondoftwo~\@gobble
362   {
363     \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
364     {
365       \lua_now:e
366       { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
367     }
368   }
369 }
370 }

```

5.1 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:
371 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
372 {
373   \int_compare:nNnTF
374     { -2147483647 }
375     =
376     {\lua_now:e
377       {
378         tex.print(tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr))
379       }
380     }
381     { \prg_return_false: }
382     { \prg_return_true: }
383   }
384
385 \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 yet if this will be global or local, see the global-mc option.

`__tag_mc_lua_set_mc_type_attr:o`

`__tag_mc_lua_unset_mc_type_attr:`

```

386 \cs_new:Nn \__tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
387 {
388   %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
389   \tl_set:Nx\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")}} }

```

```

390 \lua_now:e
391 {
392     tex.setattribute
393     (
394         "global",
395         luatexbase.attributes.g__tag_mc_type_attr,
396         \l__tag_tmpa_tl
397     )
398 }
399 \lua_now:e
400 {
401     tex.setattribute
402     (
403         "global",
404         luatexbase.attributes.g__tag_mc_cnt_attr,
405         \__tag_get_mc_abs_cnt:
406     )
407 }
408 }
409
410 \cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }
411
412 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
413 {
414     \lua_now:e
415     {
416         tex.setattribute
417         (
418             "global",
419             luatexbase.attributes.g__tag_mc_type_attr,
420             -2147483647
421         )
422     }
423     \lua_now:e
424     {
425         tex.setattribute
426         (
427             "global",
428             luatexbase.attributes.g__tag_mc_cnt_attr,
429             -2147483647
430         )
431     }
432 }
433

```

(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

```

434 \cs_new:Nn \__tag_mc_insert_mcid_kids:n
435 {
436     \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
437 }
438

```

```

439 \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
440 {
441   \lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
442 }

(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 structure.
 __tag_mc_handle_stash:o

```

443 \cs_new:Nn \__tag_mc_handle_stash:n %1 mcidnum
444 {
445   \__tag_check_mc_used:n { #1 }
446   \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
447                     % so use the kernel command
448   { g__tag_struct_kids \g__tag_struct_stack_current_tl _seq }
449   {
450     \__tag_mc_insert_mcid_kids:n {#1}%
451   }
452   \lua_now:e
453   {
454     ltx.__tag.func.store_struct_mcabs
455     (
456       \g__tag_struct_stack_current_tl,#1
457     )
458   }
459   \prop_gput:Nxx
460   \g__tag_mc_parenttree_prop
461   { #1 }
462   { \g__tag_struct_stack_current_tl }
463 }
464
465 \cs_generate_variant:Nn \__tag_mc_handle_stash:n { o }

(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.

```

466 \cs_new_protected:Nn \tag_mc_begin:n
467 {
468   \__tag_check_if_active_mc:T
469   {
470     \group_begin:
471     %\__tag_check_mc_if_nested:
472     \bool_gset_true:N \g__tag_in_mc_bool
473     \bool_set_false:N \l__tag_mc_artifact_bool
474     \tl_clear:N \l__tag_mc_key_properties_tl
475     \int_gincr:N \c@g__tag_MCID_abs_int
476     \keys_set:nn { __tag / mc }{ label={}, #1 }
477     %check that a tag or artifact has been used
478     \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
479     %set the attributes:
480     \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
481     \bool_if:NF \l__tag_mc_artifact_bool
482     { % store the absolute num name in a label:

```

```

483         \tl_if_empty:NF {\l__tag_mc_key_label_tl}
484         {
485             \exp_args:NV
486             \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
487         }
488         % if not stashed record the absolute number
489         \bool_if:NF \l__tag_mc_key_stash_bool
490         {
491             \exp_args:Nx \__tag_mc_handle_stash:n { \__tag_get_mc_abs_cnt: }
492         }
493     }
494     \group_end:
495 }
496 }

```

(End definition for `\tag_mc_begin:n`. This function is documented on page 37.)

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

```

\tag_mc_use:n 497 \cs_new_protected:Nn \tag_mc_end:
498 {
499     \__tag_check_if_active_mc:T
500     {
501         %\__tag_check_mc_if_open:
502         \bool_gset_false:N \g__tag_in_mc_bool
503         \bool_set_false:N \l__tag_mc_artifact_bool
504         \__tag_mc_lua_unset_mc_type_attr:
505         \tl_set:Nn \l__tag_mc_key_tag_tl { }
506         \tl_gset:Nn \g__tag_mc_key_tag_tl { }
507     }
508 }
509
510 \cs_new_protected:Nn \tag_mc_use:n {%#1: label name
511 {
512     \tl_set:Nx \l__tag_tmpa_tl { \__tag_ref_value:enn{tagpdf-#1}{tagmcabs}{ } }
513     \tl_if_empty:NTF \l__tag_tmpa_tl
514     {
515         \msg_warning:nnn {tag} {mc-label-unknown} { #1 }
516     }
517     {
518         \__tag_mc_handle_stash:o { \l__tag_tmpa_tl }
519     }
520 }

```

(End definition for `\tag_mc_end:` and `\tag_mc_use:n`. These functions are documented on page 37.)

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

```

521 \cs_new:Npn \__tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }

```

(End definition for `__tag_get_data_mc_tag:.`)

5.2 Key definitions

```

tag      TODO: check conversion, check if local/global setting is right.
raw      522 \keys_define:nn { __tag / mc }
alttext  523 {
alttext-o 524   tag .code:n = %
actualtext 525   {
actualtext-o 526     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
label      527     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
artifact   528     \lua_now:e
529     {
530       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
531     }
532   },
533   raw .code:n =
534   {
535     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
536     \lua_now:e
537     {
538       ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
539     }
540   },
541   alttext .code:n      = % Alt property
542   {
543     \str_set_convert:Nnon
544     \l__tag_tmpa_str
545     { #1 }
546     { default }
547     { utf16/hex }
548     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
549     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
550     \lua_now:e
551     {
552       ltx.__tag.func.store_mc_data
553       (
554         \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
555       )
556     }
557   },
558   alttext-o .code:n      = % Alt property
559   {
560     \str_set_convert:Nnon
561     \l__tag_tmpa_str
562     { #1 }
563     { default }
564     { utf16/hex }
565     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
566     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
567     \lua_now:e
568     {
569       ltx.__tag.func.store_mc_data
570       (
571         \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
572       )

```

```

573     }
574 },
575 actualtext .code:n      = % Alt property
576 {
577     \str_set_convert:Nnon
578     \l__tag_tmpa_str
579     { #1 }
580     { default }
581     { utf16/hex }
582     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt-< }
583     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
584     \lua_now:e
585     {
586         ltx.__tag.func.store_mc_data
587         (
588             \__tag_get_mc_abs_cnt:,"actualtext","/ActualText~<\str_use:N \l__tag_tmpa_str>~"
589         )
590     }
591 },
592 actualtext-o .code:n    = % Alt property
593 {
594     \str_set_convert:Noon
595     \l__tag_tmpa_str
596     { #1 }
597     { default }
598     { utf16/hex }
599     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt-< }
600     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
601     \lua_now:e
602     {
603         ltx.__tag.func.store_mc_data
604         (
605             \__tag_get_mc_abs_cnt:,
606             "actualtext",
607             "/ActualText~<\str_use:N \l__tag_tmpa_str>~"
608         )
609     }
610 },
611 label .code:n =
612 {
613     \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
614     \lua_now:e
615     {
616         ltx.__tag.func.store_mc_data
617         (
618             \__tag_get_mc_abs_cnt:,"label","#1"
619         )
620     }
621 },
622 __artifact-store .code:n =
623 {
624     \lua_now:e
625     {
626         ltx.__tag.func.store_mc_data

```

```

627         (
628             \__tag_get_mc_abs_cnt:,"artifact","#1"
629         )
630     }
631 },
632 artifact .code:n      =
633 {
634     \exp_args:Nnx
635     \keys_set:nn
636     { __tag / mc }
637     { __artifact-bool, __artifact-type=#1, tag=Artifact }
638     \exp_args:Nnx
639     \keys_set:nn
640     { __tag / mc }
641     { __artifact-store=\l__tag_mc_artifact_type_tl }
642 },
643 artifact .default:n   = { notype }
644 }
645
646 \</luamode>

```

(End definition for `tag` and others. These functions are documented on page 56.)

Part V

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><code>AF</code> = <code><object name></code> <code>AFinline</code> = <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} {2021-06-14} {0.82}
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 `\l__tag_struct_stack_parent_tmpa_tl` 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

```

      \tag_struct_begin:n { tag=Link }
      \tag_mc_begin:n { tag=Link }
(1)    \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_0000lang
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 57.)

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
411 ,AFinline .code:n =
412 {
413   \group_begin:
414   \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
415   {
416     \pdffile_embed_stream:nxx
417     {#1}
418     {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
419     {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
420   }
421   \__tag_prop_gput:cnx
422   { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
423   { AF }
424   { \pdf_object_ref:e {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int} } }
425   \group_end:
426 }
427 ,AFinline-o .code:n =
428 {
429   \group_begin:
430   \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
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 57.)

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       \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
514       {
515         \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
516       }
517     }
518     { \__tag_check_no_open_struct: }
519     % get the previous one, shouldn't be empty as the root should be there
520     \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
521     {
522       \tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
523     }
524     {
525       \__tag_check_no_open_struct:
526     }
527     \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
528     {
529       \tl_gset:NV \g__tag_struct_tag_tl \l__tag_tmpa_tl
530     }
531   }
532 }

```

(End definition for `\tag_struct_begin:n` and `\tag_struct_end:.` These functions are documented on page 56.)

`\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.

```

533 \cs_new_protected:Nn \tag_struct_use:n {%#1 is the label
534 {
535   \__tag_check_if_active_struct:T
536   {
537     \prop_if_exist:cTF
538     { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
539     {
540       \__tag_check_struct_used:n {#1}
541       %add the label structure as kid to the current structure (can be the root)
542       \__tag_struct_kid_struct_gput_right:xx
543       { \g__tag_struct_stack_current_tl }
544       { \__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0} }
545       %add the current structure to the labeled one as parents
546       \__tag_prop_gput:cnx
547       { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
548       { P }
549       {
550         \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
551       }
552     }
553     {
554       \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
555     }
556   }
557 }

```

(End definition for `\tag_struct_use:n`. This function is documented on page 56.)

`\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.

```

558 \cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 {%#1 should be an object reference
559                                     %#2 struct parent num
560 {
561   \__tag_check_if_active_struct:T
562   {
563     \__tag_struct_insert_annot:nn {#1}{#2}
564   }
565 }
566
567 \cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx}
568 \cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}
569
570 \</package>
571

```

(End definition for `\tag_struct_insert_annot:nn` and `\tag_struct_parent_int:.` These functions are documented on page 56.)

7 Attributes and attribute classes

```

572 \<header>
573 \ProvidesExplPackage {tagpdf-attr-code} {2021-06-14} {0.82}
574 {part of tagpdf - code related to attributes and attribute classes}
575 \</header>

```

7.1 Variables

`\g__tag_attr_entries_prop` `\g__@@_attr_entries_prop` will store attribute names and their dictionary content.
`\g__tag_attr_class_used_seq` `\g__@@_attr_class_used_seq` will hold the attributes which have been used as class name.
`\g__tag_attr_objref_prop` `\l__@@_attr_value_tl` is used to build the attribute array or key. 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`
`\l__tag_attr_value_tl`

```

576 \<*package>
577 \prop_new:N \g__tag_attr_entries_prop
578 \seq_new:N \g__tag_attr_class_used_seq
579 \tl_new:N \l__tag_attr_value_tl
580 \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},
}

581 \cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
582 {
583   \prop_gput:Nnn \g__tag_attr_entries_prop
584     {#1}{#2}
585 }
586
587 \keys_define:nn { __tag / setup }
588 {
589   newattribute .code:n =
590     {
591       \__tag_attr_new_entry:nn #1
592     }
593 }

```

(End definition for `__tag_attr_new_entry:nn` and `newattribute`. This function is documented on page 58.)

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

```

594 \keys_define:nn { __tag / struct }
595 {
596   attribute-class .code:n =
597   {
598     \clist_set:No \l__tag_tmpa_clist { #1 }
599     \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
600     \seq_map_inline:Nn \l__tag_tmpa_seq
601     {
602       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
603       {
604         \msg_error:nnn { tag } { attr-unknown } { ##1 }
605       }
606       \seq_gput_left:Nn\g__tag_attr_class_used_seq { ##1}
607     }
608     \seq_set_map:NNn \l__tag_tmpb_seq \l__tag_tmpa_seq
609     {
610       /##1
611     }
612     \tl_set:Nx \l__tag_tmpa_tl
613     {
614       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
615       \seq_use:Nn \l__tag_tmpb_seq { \c_space_tl }
616       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
617     }
618     \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
619     {
620       __tag_prop_gput:cnx
621       { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
622       { C }
623       { \l__tag_tmpa_tl }
624       %\prop_show:c { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
625     }
626   }
627 }

```

(End definition for attribute-class. This function is documented on page 58.)

attribute

```

628 \keys_define:nn { __tag / struct }
629 {
630   attribute .code:n = % A property (attribute, value currently a dictionary)
631   {
632     \clist_set:No \l__tag_tmpa_clist { #1 }
633     \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
634     \tl_set:Nx \l__tag_attr_value_tl
635     {
636       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[]}
637     }
638     \seq_map_inline:Nn \l__tag_tmpa_seq
639     {
640       \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
641       {

```

```

642         \msg_error:nnn { tag } { attr-unknown } { ##1 }
643     }
644     \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
645     {%\prop_show:N \g__tag_attr_entries_prop
646       \pdf_object_unnamed_write:nx
647       { dict }
648       {
649         \prop_item:Nn\g__tag_attr_entries_prop {##1}
650       }
651       \prop_gput:Nnx \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
652     }
653     \tl_put_right:Nx \l__tag_attr_value_tl
654     {
655       \c_space_tl
656       \prop_item:Nn \g__tag_attr_objref_prop {##1}
657     }
658     % \tl_show:N \l__tag_attr_value_tl
659     }
660     \tl_put_right:Nx \l__tag_attr_value_tl
661     { %[
662       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
663     }
664     % \tl_show:N \l__tag_attr_value_tl
665     \__tag_prop_gput:cnx
666     { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
667     { A }
668     { \l__tag_attr_value_tl }
669   },
670 }
671 \end{package}

```

(End definition for attribute. This function is documented on page 58.)

Part VI

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

```
1 <@@=tag>
2 <*luatex>
3 \ProvidesExplFile {tagpdf-luatex.def} {2021-06-14} {0.82}
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 is probably unneeded and should be cleaned up!

```

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

```

30 \cs_set_protected:Nn \__tag_seq_gput_right:Nn
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:Nn \__tag_seq_show:N
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:Nn \__tag_prop_show:N
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 }
62
63 (End definition for \__tag_prop_new:N and others.)
64
65 \end{luatex}
66
67 \begin{lua}
68 -- tagpdf.lua
69 -- Ulrike Fischer
70
71 local ProvidesLuaModule = {
72   name      = "tagpdf",
73   version   = "0.82",      --TAGVERSION
74   date      = "2021-06-14", --TAGDATE
75   description = "tagpdf lua code",
76   license    = "The LATEX Project Public License 1.3c"
77 }
78
79 if luatexbase and luatexbase.provides_module then
80   luatexbase.provides_module (ProvidesLuaModule)
81 end
82
83 --[[
84 The code has quite probably a number of problems
85 - 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
88
89
90 --[[
91 the main table is named ltx.__tag. It contains the functions and also the data
92 collected during the compilation.
93
94 ltx.__tag.mc      will contain mc connected data.
95 ltx.__tag.struct  will contain structure related data.
96 ltx.__tag.page    will contain page data
97 ltx.__tag.tables  contains also data from mc and struct (from older code). This needs cleaning
98                 There are certainly dublettes, but I don't dare yet ...
99 ltx.__tag.func    will contain (public) functions.
100 ltx.__tag.trace   will contain tracing/logging functions.
101 local funktions starts with __
102 functions meant for users will be in ltx.tag
103
104 functions
105 ltx.__tag.func.get_num_from (tag):   takes a tag (string) and returns the id number
106 ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
107 ltx.__tag.func.get_tag_from (num):   takes a num and returns the tag
108 ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
109 ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
110 ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
111 ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
112 ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number of
113 ltx.__tag.func.store_struct_mcab (structnum,mcnum): stores relations structnum<->mcnum (abs)
114 ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
115 ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
116 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EM
117 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this p
118 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
119 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of pos
120 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
121 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
122 ltx.__tag.trace.show_seq: shows a sequence (array)
123 ltx.__tag.trace.show_struct_data (num): shows data of structure num
124 ltx.__tag.trace.show_prop: shows a prop
125 ltx.__tag.trace.log
126 ltx.__tag.trace.showspace : boolean
127 --]]
128
129
130 local mctypeattributeid = luatexbase.new_attribute ("g__tag_mc_type_attr")
131 local mcntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
132 local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
133 local iwfontattributeid  = luatexbase.new_attribute ("g__tag_interwordfont_attr")
134
135 local catlatex          = luatexbase.registernumber("catcodetable@latex")

```

```

136 local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
137 local truebool      = token.create("c_true_bool")
138
139 local tableinsert    = table.insert
140
141 -- not all needed, copied from lua-visual-debug.
142 local nodeid         = node.id
143 local nodecopy       = node.copy
144 local nodegetattribute = node.get_attribute
145 local nodesetattribute = node.set_attribute
146 local nodehasattribute = node.has_attribute
147 local nodenew       = node.new
148 local nodetail      = node.tail
149 local nodeslide     = node.slide
150 local noderemove    = node.remove
151 local nodetraverseid = node.traverse_id
152 local nodetraverse  = node.traverse
153 local nodeinsertafter = node.insert_after
154 local nodeinsertbefore = node.insert_before
155 local pdfpageref    = pdf.pageref
156
157 local HLIST         = node.id("hlist")
158 local VLIST         = node.id("vlist")
159 local RULE          = node.id("rule")
160 local DISC          = node.id("disc")
161 local GLUE          = node.id("glue")
162 local GLYPH         = node.id("glyph")
163 local KERN          = node.id("kern")
164 local PENALTY       = node.id("penalty")
165 local LOCAL_PAR     = node.id("local_par")
166 local MATH          = node.id("math")
167
168 local function __tag_get_mathsubtype (mathnode)
169   if mathnode.subtype == 0 then
170     subtype = "beginmath"
171   else
172     subtype = "endmath"
173   end
174   return subtype
175 end
176
177
178
179 ltx          = ltx          or { }
180 ltx.__tag    = ltx.__tag    or { }
181 ltx.__tag.mc = ltx.__tag.mc  or { } -- mc data
182 ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
183 ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new s
184                                     -- wasn't a so great idea ...
185 ltx.__tag.page = ltx.__tag.page or { } -- page data, currently only i->{0->mcnum
186 ltx.__tag.trace = ltx.__tag.trace or { } -- show commands
187 ltx.__tag.func  = ltx.__tag.func  or { } -- functions
188 ltx.__tag.conf  = ltx.__tag.conf  or { } -- configuration variables
189

```

```

190 local function fakespace()
191     tex.setattribute(iwspaceattributeid,1)
192     tex.setattribute(iwfontattributeid,font.current())
193 end
194 ltx.__tag.func.fakespace = fakespace
195
196
197 local __tag_log =
198     function (message,loglevel)
199         if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
200             texio.write_nl("tagpdf: ".. message)
201         end
202     end
203
204 ltx.__tag.trace.log = __tag_log
205
206
207 local __tag_get_mc_cnt_type_tag = function (n)
208     local mccnt      = nodegetattribute(n,mccntattributeid) or -1
209     local mctype     = nodegetattribute(n,mctypeattributeid) or -1
210     local tag        = ltx.__tag.func.get_tag_from(mctype)
211     return mccnt,mctype,tag
212 end
213
214
215 local function __tag_insert_emc_node (head,current)
216     local emcnode = nodenew("whatsit","pdf_literal")
217     emcnode.data = "EMC"
218     emcnode.mode=1
219     head = node.insert_before(head,current,emcnode)
220     return head
221 end
222
223
224 local function __tag_insert_bmc_node (head,current,tag)
225     local bmcnode = nodenew("whatsit","pdf_literal")
226     bmcnode.data = "/"..tag.." BMC"
227     bmcnode.mode=1
228     head = node.insert_before(head,current,bmcnode)
229     return head
230 end
231
232 local function __tag_insert_bdc_node (head,current,tag,dict)
233     local bdcnode = nodenew("whatsit","pdf_literal")
234     bdcnode.data = "/"..tag.."<<..dict..>> BDC"
235     bdcnode.mode=1
236     head = node.insert_before(head,current,bdcnode)
237     return head
238 end
239
240 local function __tag_pdf_object_ref (name)
241     local tokenname = 'c__pdf_backend_object_'..name..'_int'
242     local object = token.create(tokenname).index..' 0 R'
243     return object

```



```

244 end
245 ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref
246
247 -- this is for debugging the space chars
248 local function __tag_show_spacemark (head,current,color,height)
249     local markcolor = color or "1 0 0"
250     local markheight = height or 10
251     local pdfstring = node.new("whatsit","pdf_literal")
252     pdfstring.data =
253         string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
254             3,markheight)
255     head = node.insert_after(head,current,pdfstring)
256     return head
257 end
258
259 --[[ a function to mark up places where real space chars should be inserted
260     it only sets an attribute.
261 --]]
262
263 local function __tag_mark_spaces (head)
264     local inside_math = false
265     for n in nodetraverse(head) do
266         local id = n.id
267         if id == GLYPH then
268             local glyph = n
269             if glyph.next and (glyph.next.id == GLUE)
270                 and not inside_math and (glyph.next.width > 0)
271             then
272                 nodesetattribute(glyph.next,iwspaceattributeid,1)
273                 nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
274             -- for debugging
275             if ltx.__tag.trace.showspace then
276                 __tag_show_spacemark (head,glyph)
277             end
278             elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
279                 local kern = glyph.next
280                 if kern.next and (kern.next.id== GLUE) and (kern.next.width > 0)
281                 then
282                     nodesetattribute(kern.next,iwspaceattributeid,1)
283                     nodesetattribute(kern.next,iwfontattributeid,glyph.font)
284                 end
285             end
286             -- look also back
287             if glyph.prev and (glyph.prev.id == GLUE)
288                 and not inside_math and (glyph.prev.width > 0) and not nodehasattribute(glyph.prev,iw
289             then
290                 nodesetattribute(glyph.prev,iwspaceattributeid,1)
291                 nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
292             -- for debugging
293             if ltx.__tag.trace.showspace then
294                 __tag_show_spacemark (head,glyph)
295             end
296             end
297             elseif id == PENALTY then

```

```

297     local glyph = n
298     -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
299     if glyph.next and (glyph.next.id == GLUE)
300       and not inside_math and (glyph.next.width > 0) and n.subtype==0
301     then
302       nodesetattribute(glyph.next,iwspaceattributeid,1)
303       -- nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
304       -- for debugging
305       if ltx.__tag.trace.showspace then
306         __tag_show_spacemark (head,glyph)
307       end
308     end
309     elseif id == MATH then
310       inside_math = (n.subtype == 0)
311     end
312   end
313   return head
314 end
315
316 local function __tag_activate_mark_space ()
317   if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
318     luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
319     luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
320   end
321 end
322
323 ltx.__tag.func.markspaceon=__tag_activate_mark_space
324
325 local function __tag_deactivate_mark_space ()
326   if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
327     luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
328     luatexbase.remove_from_callback("hpack_filter","markspaces")
329   end
330 end
331 --
332 ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
333
334 local default_space_char = node.new(GLYPH)
335 local default_fontid      = font.id("TU/lmr/m/n/10")
336 default_space_char.char    = 32
337 default_space_char.font    = default_fontid
338
339 local function __tag_insert_space_char (head,n,fontid)
340   if luaotfload.aux.slot_of_name(fontid,"space") then
341     local space
342     -- head, space = node.insert_before(head, n, ) -- Set the right font
343     -- n.width = n.width - space.width
344     -- space.attr = n.attr
345   end
346 end
347
348 --[[
349     Now follows the core function
350     It wades through the shipout box and checks the attributes

```

```

351 ARGUMENTS
352 box: is a box,
353 mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
354 mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a c
355 mcopen: num, records if some bdc/emc is open
356 These arguments are only needed for log messages, if not present are replaces by fix string
357 name: string to describe the box
358 mctypeprev: num, the type attribute of the previous node/whatever
359
360 there are lots of logging messages currently. Should be cleaned up in due course.
361 One should also find ways to make the function shorter.
362 --]]
363
364 function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
365     local name = name or ("SOMEBBOX")
366     local mctypeprev = mctypeprev or -1
367     local abspage = status.total_pages + 1 -- the real counter is increased inside the box so c
368                                           -- if the callback is r
369     ltx.__tag.trace.log ("PAGE " .. abspage,3)
370     ltx.__tag.trace.log ("FUNC ARGS: pagecnt".. mcpagecnt.." prev "..mccntprev .. " type prev "
371     ltx.__tag.trace.log ("TRAVERSING BOX ".. tostring(name).." TYPE ".. node.type(node.getid(box
372     local head = box.head -- ShipoutBox is a vlist?
373     if head then
374         mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
375         ltx.__tag.trace.log ("HEAD " .. node.type(node.getid(head)).. " MC"..tostring(mccnthead).
376     else
377         ltx.__tag.trace.log ("HEAD is ".. tostring(head),3)
378     end
379     for n in node.traverse(head) do
380         local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
381         local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
382         ltx.__tag.trace.log ("NODE " .. node.type(node.getid(n)).." MC"..tostring(mccnt).. => TAG
383         if n.id == HLIST
384         then -- enter the hlist
385             mcopen,mcpagecnt,mccntprev,mctypeprev=
386             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypeprev
387         elseif n.id == VLIST then -- enter the vlist
388             mcopen,mcpagecnt,mccntprev,mctypeprev=
389             ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypeprev
390         elseif n.id == GLUE then -- at glue real space chars are inserted, for the rest it i
391             -- for debugging
392             if ltx.__tag.trace.showspace and spaceattr==1 then
393                 __tag_show_spacemark (head,n,"0 1 0")
394             end
395             if spaceattr==1 then
396                 local space
397                 local space_char = node.copy(default_space_char)
398                 local curfont = nodegetattribute(n,iwfontattributeid)
399                 ltx.__tag.trace.log ("FONT " .. tostring(curfont),3)
400                 if curfont and luaotfload.aux.slot_of_name(curfont,"space") then
401                     space_char.font=curfont
402                 end
403                 head, space = node.insert_before(head, n, space_char) --
404                 n.width = n.width - space.width

```

```

405     space.attr = n.attr
406 end
407 elseif n.id == LOCAL_PAR then -- local_par is ignored
408 elseif n.id == PENALTY then -- penalty is ignored
409 elseif n.id == KERN then -- kern is ignored
410     ltx.__tag.trace.log ("SUBTYPE KERN ".. n.subtype,3)
411 else
412     -- math is currently only logged.
413     -- we could mark the whole as math
414     -- for inner processing the mlist_to_hlist callback is probably needed.
415     if n.id == MATH then
416         ltx.__tag.trace.log("NODE "..node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),3)
417     end
418     -- endmath
419     ltx.__tag.trace.log("CURRENT "..mccnt.." PREV "..mccntprev,3)
420     if mccnt~=mccntprev then -- a new mc chunk
421         ltx.__tag.trace.log ("NODE ".. node.type(node.getid(n)).." MC"..tostring(mccnt).." <=> I")
422         if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
423             box.list=__tag_insert_emc_node (box.list,n)
424             mcopen = mcopen - 1
425             ltx.__tag.trace.log ("INSERT EMC " .. mcpagencnt .. " MCOPEN = " .. mcopen,2)
426             if mcopen ~=0 then
427                 ltx.__tag.trace.log ("!WARNING! open mc" .. " MCOPEN = " .. mcopen,1)
428             end
429         end
430         if ltx.__tag.mc[mccnt] then
431             if ltx.__tag.mc[mccnt]["artifact"] then
432                 ltx.__tag.trace.log("THIS IS AN ARTIFACT of type "..tostring(ltx.__tag.mc[mccnt]["art
433                 if ltx.__tag.mc[mccnt]["artifact"] == "" then
434                     box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
435                 else
436                     box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]
437                 end
438             else
439                 ltx.__tag.trace.log("THIS IS A TAG "..tostring(tag),3)
440                 mcpagencnt = mcpagencnt +1
441                 ltx.__tag.trace.log ("INSERT BDC " ..mcpagencnt,2)
442                 local dict= "/MCID "..mcpagencnt
443                 if ltx.__tag.mc[mccnt]["raw"] then
444                     ltx.__tag.trace.log("RAW CONTENT"..tostring(ltx.__tag.mc[mccnt]["raw"]),3)
445                     dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
446                 end
447                 if ltx.__tag.mc[mccnt]["alt"] then
448                     ltx.__tag.trace.log("RAW CONTENT"..tostring(ltx.__tag.mc[mccnt]["alt"]),3)
449                     dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
450                 end
451                 if ltx.__tag.mc[mccnt]["actualtext"] then
452                     ltx.__tag.trace.log("RAW CONTENT"..tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
453                     dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
454                 end
455                 box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
456                 ltx.__tag.func.store_mc_kid (mccnt,mcpagencnt,abspage)
457                 ltx.__tag.func.store_mc_in_page(mccnt,mcpagencnt,abspage)
458                 ltx.__tag.trace.show_mc_data (mccnt,3)

```

```

459         end
460         mcopen = mcopen + 1
461     else
462         ltx.__tag.trace.log("THIS HAS NOT BEEN TAGGED",1)
463     -- perhaps code that tag a artifact can be added ...
464         if tagunmarkedbool.mode == truebool.mode then
465             box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
466             mcopen = mcopen + 1
467         end
468     end
469     mccntprev = mccnt
470     end
471 end -- end if
472 end -- end for
473 if head then
474     mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
475     ltx.__tag.trace.log ("ENDHEAD " .. node.type(node.getid(head)).. " MC"..tostring(mccnthead)
476 else
477     ltx.__tag.trace.log ("ENDHEAD is " .. tostring(head),3)
478 end
479 ltx.__tag.trace.log ("QUITTING TRAVERSING BOX " .. tostring(name).. " TYPE " .. node.type(node.getid(head)))
480 return mcopen,mcpagecnt,mccntprev,mctypeprev
481 end
482
483
484
485 function ltx.__tag.func.mark_shipout (box)
486     mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
487     if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
488         local emcnode = nodenew("whatsit","pdf_literal")
489         local list = box.list
490         emcnode.data = "EMC"
491         emcnode.mode=1
492         if list then
493             list = node.insert_after (list,node.tail(list),emcnode)
494             mcopen = mcopen - 1
495             ltx.__tag.trace.log ("INSERT LAST EMC, MCOPEN = " .. mcopen,2)
496         else
497             ltx.__tag.trace.log ("UPS ",1)
498         end
499         if mcopen ~=0 then
500             ltx.__tag.trace.log ("!WARNING! open mc" .. " MCOPEN = " .. mcopen,1)
501         end
502     end
503 end
504
505
506 function ltx.__tag.trace.show_seq (seq)
507     if (type(seq) == "table") then
508         for i,v in ipairs(seq) do
509             __tag_log ("[" .. i .. "] => " .. tostring(v),1)
510         end
511     else
512         __tag_log ("sequence " .. tostring(seq) .. " not found",1)

```

```

513     end
514 end
515
516 local __tag_pairs_prop =
517 function (prop)
518     local a = {}
519     for n in pairs(prop) do tableinsert(a, n) end
520     table.sort(a)
521     local i = 0                -- iterator variable
522     local iter = function ()   -- iterator function
523         i = i + 1
524         if a[i] == nil then return nil
525         else return a[i], prop[a[i]]
526         end
527     end
528     return iter
529 end
530
531
532 function ltx.__tag.trace.show_prop (prop)
533 if (type(prop) == "table") then
534     for i,v in __tag_pairs_prop (prop) do
535         __tag_log ("[" .. i .. "] => " .. tostring(v),1)
536     end
537 else
538     __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
539 end
540 end
541
542
543 local __tag_get_num_from =
544 function (tag)
545     if ltx.__tag.tables["g__tag_role_tags_prop"][tag] then
546         a = ltx.__tag.tables["g__tag_role_tags_prop"][tag]
547     else
548         a = -1
549     end
550     return a
551 end
552
553 ltx.__tag.func.get_num_from = __tag_get_num_from
554
555 function ltx.__tag.func.output_num_from (tag)
556     local num = __tag_get_num_from (tag)
557     tex.sprint(catlatex,num)
558     if num == -1 then
559         __tag_log ("Unknown tag "..tag.." used")
560     end
561 end
562
563 local __tag_get_tag_from =
564 function (num)
565     if ltx.__tag.tables["g__tag_role_tags_seq"][num] then
566         a = ltx.__tag.tables["g__tag_role_tags_seq"][num]

```

```

567     else
568         a= "UNKNOWN"
569     end
570     return a
571 end
572
573 ltx.__tag.func.get_tag_from = __tag_get_tag_from
574
575 function ltx.__tag.func.output_tag_from (num)
576     tex.sprint(catlatex,__tag_get_tag_from (num))
577 end
578
579 function ltx.__tag.func.store_mc_data (num,key,data)
580     ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
581     ltx.__tag.mc[num][key] = data
582     __tag_log ("storing mc"..num..": "..tostring(key)..">"..tostring(data))
583 end
584
585 function ltx.__tag.trace.show_mc_data (num,loglevel)
586     if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
587         for k,v in pairs(ltx.__tag.mc[num]) do
588             __tag_log ("mc"..num..": "..tostring(k)..">"..tostring(v),loglevel)
589         end
590         if ltx.__tag.mc[num]["kids"] then
591             __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
592             for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
593                 __tag_log ("mc " .. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
594             end
595         end
596     else
597         __tag_log ("mc"..num.." not found",3)
598     end
599 end
600
601 function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
602     for i = min, max do
603         ltx.__tag.trace.show_mc_data (i,loglevel)
604     end
605     texio.write_nl("")
606 end
607
608
609 function ltx.__tag.func.store_mc_label (label,num)
610     ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
611     ltx.__tag.mc.labels[label] = num
612 end
613
614 function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
615     ltx.__tag.trace.log("MC"..mcnum.." STORING KID" .. kid.." on page " .. page,3)
616     ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
617     local kidtable = {kid=kid,page=page}
618     tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
619 end
620

```

```

621
622 function ltx.__tag.func.mc_num_of_kids (mcnum)
623   local num = 0
624   if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
625     num = #ltx.__tag.mc[mcnum]["kids"]
626   end
627   ltx.__tag.trace.log ("MC" .. mcnum .. "has " .. num .. "KIDS",4)
628   return num
629 end
630
631 function ltx.__tag.func.mc_insert_kids (mcnum,single)
632   if ltx.__tag.mc[mcnum] then
633     ltx.__tag.trace.log("MC-KIDS test " .. mcnum,4)
634     if ltx.__tag.mc[mcnum]["kids"] then
635       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
636         tex.sprint("")
637       end
638       for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
639         local kidnum = kidstable["kid"]
640         local kidpage = kidstable["page"]
641         local kidpageobjnum = pdfpageref(kidpage)
642         ltx.__tag.trace.log("MC" .. mcnum .. " insert KID " .. i.. " with num " .. kidnum .. " on
643         tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> " )
644       end
645       if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
646         tex.sprint("")
647       end
648     else
649       -- this is typically not a problem, e.g. empty hbox in footer/header can
650       -- trigger this warning.
651       ltx.__tag.trace.log("WARN! MC"..mcnum.." has no kids",2)
652       if single==1 then
653         tex.sprint("null")
654       end
655     end
656   else
657     ltx.__tag.trace.log("WARN! MC"..mcnum.." doesn't exist",0)
658   end
659 end
660
661
662 function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
663   ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
664   ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
665   -- a structure can contain more than one mc chunk, the content should be ordered
666   tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
667   ltx.__tag.trace.log("MCNUM "..mcnum.." insert in struct "..structnum,3)
668   -- but every mc can only be in one structure
669   ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
670   ltx.__tag.mc[mcnum]["parent"] = structnum
671 end
672
673 function ltx.__tag.trace.show_struct_data (num)
674   if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then

```



```

675     for k,v in ipairs(ltx.__tag.struct[num]) do
676         __tag_log ("struct "..num..": "..tostring(k).."=>"..tostring(v))
677     end
678 else
679     __tag_log ("struct "..num.." not found ")
680 end
681 end
682
683 -- pay attention: lua counts arrays from 1, tex pages from one
684 -- mcid and arrays in pdf count from 0.
685 function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
686     ltx.__tag.page[page] = ltx.__tag.page[page] or {}
687     ltx.__tag.page[page][mcpagecnt] = mcnum
688     ltx.__tag.trace.log("PAGE " .. page .. ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
689 end
690
691 function ltx.__tag.func.fill_parent_tree_line (page)
692     -- we need to get page-> i=kid -> mcnum -> structnum
693     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
694     local numsentry = ""
695     local pdfpage = page-1
696     if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
697         mcchunks=#ltx.__tag.page[page]
698         ltx.__tag.trace.log("PAGETREE PAGE "..page.." has "..mcchunks.."+1 Elements ",3)
699         for i=0,mcchunks do
700             ltx.__tag.trace.log("PAGETREE CHUNKS "..ltx.__tag.page[page][i],3)
701         end
702         if mcchunks == 0 then
703             -- only one chunk so no need for an array
704             local mcnum = ltx.__tag.page[page][0]
705             local structnum = ltx.__tag.mc[mcnum]["parent"]
706             local propname = "g__tag_struct_"..structnum.."__prop"
707             --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
708             local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
709             ltx.__tag.trace.log("====>"..tostring(objref),5)
710             numsentry = pdfpage .. " [".. objref .. "]"
711             ltx.__tag.trace.log("PAGETREE PAGE" .. page.. " NUM ENTRY = ".. numsentry,3)
712         else
713             numsentry = pdfpage .. " ["
714             for i=0,mcchunks do
715                 local mcnum = ltx.__tag.page[page][i]
716                 local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
717                 local propname = "g__tag_struct_"..structnum.."__prop"
718                 --local objref = ltx.__tag.tables[propname]["objref"] or "XXXX"
719                 local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
720                 numsentry = numsentry .. " " .. objref
721             end
722             numsentry = numsentry .. "]"
723             ltx.__tag.trace.log("PAGETREE PAGE" .. page.. " NUM ENTRY = ".. numsentry,3)
724         end
725     else
726         ltx.__tag.trace.log ("PAGETREE: NO DATA FOR PAGE "..page,3)
727     end
728     return numsentry

```

```

729 end
730
731 function ltx.__tag.func.output_parenttree (abspage)
732   for i=1,abspage do
733     line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
734     tex.sprint(catlatex,line)
735   end
736 end
737
738
739  $\langle$ /lua)

```

Part VII

The tagpdf-roles module

Tags, roles and namespace code part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-roles-code} {2021-06-14} {0.82}
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   \msg_redirect_name:nnn { pdfdict } { empty-value } { none }
18   \pdf_object_new:nn {tag/NS/#1}{dict}
19   \pdfdict_new:n {g__tag_role/Namespace_#1_dict}
20   \pdf_object_new:nn {\__tag/RoleMapNS/#1}{dict}
21   \pdfdict_new:n {g__tag_role/RoleMapNS_#1_dict}
22   \pdfdict_gput:nnn
23     {g__tag_role/Namespace_#1_dict}
24     {Type}
25     {/Namespace}
26   \pdf_string_from_unicode:nnN{utf8/string}{#2}\l_tmpa_str
27   \pdfdict_gput:nnx
28     {g__tag_role/Namespace_#1_dict}
29     {NS}
30     {\l_tmpa_str}
31   %RoleMapNS is added in tree
32   \pdfdict_gput:nnx{g__tag_role/Namespace_#1_dict}
33     {Schema}{#3}
34   \prop_gput:Nnx \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
35   \msg_redirect_name:nnn { pdfdict } { empty-value } { warning }
36 }
```

(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

```

37 \str_const:Nx \c__tag_role_userNS_id_str
38 { data:,
39   \int_to_Hex:n{\int_rand:n {65535}}
40   \int_to_Hex:n{\int_rand:n {65535}}
41   -
42   \int_to_Hex:n{\int_rand:n {65535}}
43   -
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 {16777215}}
49   \int_to_Hex:n{\int_rand:n {16777215}}
50 }

```

(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.

```

51 \pdf_version_compare:NnT > {1.9}
52 {
53   \__tag_role_NS_new:nnn {pdf}   {http://iso.org/pdf/ssn}{ }
54   \__tag_role_NS_new:nnn {pdf2}  {http://iso.org/pdf2/ssn}{ }
55   \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{ }
56   \exp_args:Nnx
57   \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{ }
58 }

```

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
59 %
60 \clist_const:Nn \c__tag_role_sttags_pdf_pdfII_clist
61 {
62   Document,    %A complete document. This is the root element
63                %of any structure tree containing
64                %multiple parts or multiple articles.
65   Part,        %A large-scale division of a document.
66   Sect,        %A container for grouping related content elements.
67   Div,         %A generic block-level element or group of elements
68   Caption,     %A brief portion of text describing a table or figure.
69   Index,
70   NonStruct,   %probably not needed
71   H,
72   H1,
73   H2,
74   H3,
75   H4,

```

```

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

```

```

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

```

```

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

```


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

```

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

```

```

346     H8      = H6 ,
347     H9      = H6 ,
348     H10     = H6,
349     FENote= Note,
350     Em      = Span,
351     Strong= Span,
352 }

```

(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.

```

353 % get tag name from number: \seq_item:Nn \g__tag_role_tags_seq { n }
354 % get tag number from name: \prop_item:Nn \g__tag_role_tags_prop { name }
355
356 \clist_map_inline:Nn \c__tag_role_sttags_pdf_pdfII_clist
357 {
358     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
359     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf2 }
360 }
361 \clist_map_inline:Nn \c__tag_role_sttags_only_pdf_clist
362 {
363     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
364     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf }
365 }
366 \clist_map_inline:Nn \c__tag_role_sttags_only_pdfII_clist
367 {
368     \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
369     \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ pdf2 }
370 }
371 \pdf_version_compare:NnT > {1.9}
372 {
373     \clist_map_inline:Nn \c__tag_role_sttags_mathml_clist
374     {
375         \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
376         \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ mathml }
377     }
378 }

```

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

```

379 \int_step_inline:nnnn { 1 }{ 1 }{ \seq_count:N \g__tag_role_tags_seq }
380 {
381     \__tag_prop_gput:Nxn \g__tag_role_tags_prop
382     {
383         \seq_item:Nn \g__tag_role_tags_seq { #1 }
384     }
385     { #1 }
386 }

```

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

```
387 \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.

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

```
389 {
```

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

```
391   {
```

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

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

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

```
395     {
```

```
396       \seq_count:N \g__tag_role_tags_seq
```

```
397     }
```

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

```
399   }
```

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

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

```
402   {
```

```
403     \pdfdict_gput:nxx {g__tag_role/RoleMap_dict}
```

```
404     {#1}
```

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

```
406   }
```

```
407 }
```

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

```
409
```

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

```
411 {
```

```
412   \prop_map_inline:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
```

```
413   {
```

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

```
415   }
```

```
416 }
```

```
417
```

(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

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

```
419 {
```

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

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

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

```
423   {
```

```
424     \seq_count:N \g__tag_role_tags_seq
```

```
425   }
```

```
426   \prop_gput:Nnn \g__tag_role_tags_NS_prop { #1 }{ #2 }
```

```
427   \__tag_check_add_tag_role:nn {#1}{#3}
```

```
428   \pdfdict_gput:nxx {g__tag_role/RoleMapNS_#2_dict}{#1}
```

```

429     {
430     [
431         \pdf_name_from_unicode_e:n{#3}
432         \c_space_tl
433         \pdf_object_ref:n {tag/NS/#4}
434     ]
435     }
436 }
437 \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 438 \keys_define:nn { __tag / tag-role }
role           439 {
role-namespace 440     ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
add-new-tag    441     ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
               442     ,role .tl_set:N = \l__tag_role_role_tmpa_tl
               443     ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
               444     }
               445
               446 \keys_define:nn { __tag / setup }
               447 {
               448     add-new-tag .code:n =
               449     {
               450         \keys_set_known:nnnN
               451         {__tag/tag-role}
               452         {
               453             tag-namespace=user,
               454             role-namespace=, %so that we can test for it.
               455             #1
               456             }{__tag/tag-role}\l_tmpa_tl
               457         \tl_if_empty:NF \l_tmpa_tl
               458         {
               459             \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
               460             \tl_set:Nx \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} }
               461             \tl_set:Nx \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
               462         }
               463         \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
               464         {
               465             \prop_get:NVNTF
               466             \g__tag_role_tags_NS_prop
               467             \l__tag_role_role_tmpa_tl
               468             \l__tag_role_role_namespace_tmpa_tl
               469             {
               470                 \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
               471                 {
               472                     \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
               473                 }

```

```

474         }
475         {
476             \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
477         }
478     }
479     \pdf_version_compare:NnTF < {2.0}
480     {
481         %TODO add check for emptyness?
482         \__tag_role_add_tag:VV
483             \l__tag_role_tag_tmpa_tl
484             \l__tag_role_role_tmpa_tl
485     }
486     {
487         \__tag_role_add_tag:VVVV
488             \l__tag_role_tag_tmpa_tl
489             \l__tag_role_tag_namespace_tmpa_tl
490             \l__tag_role_role_tmpa_tl
491             \l__tag_role_role_namespace_tmpa_tl
492     }
493 }
494 }
495 \end{package}

```

(End definition for tag and others. These functions are documented on page 56.)

Part VIII

The tagpdf-space module code related to real space chars part of the tagpdf package

```
1 <@@=tag>
2 <*header>
3 \ProvidesExplPackage {tagpdf-space-code} {2021-06-14} {0.82}
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       show-spaces .bool_set:N = \l__tag_showspaces_bool
17     }
18   }
19   {
20     \keys_define:nn { __tag / setup }
21     {
22       interwordspace .choices:nn = { true, on, false, off }
23       { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi} },
24       show-spaces .bool_set:N = \l__tag_showspaces_bool
25     }
26   }
27 }
28
29
30 \sys_if_engine_luaTeX:T
31 {
32   \keys_define:nn { __tag / setup }
33   {
34     interwordspace .choices:nn =
35     { true, on }
36     { \lua_now:e{!tx.__tag.func.markspaceon()} },
37     interwordspace .choices:nn =
```

```

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

```

(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.

```

60 \sys_if_engine_luatex:T
61 {
62   \cs_new_protected:Nn \__tag_fakespace:
63   {
64     \group_begin:
65     %\lua_now:e{tex.setattribute("g__tag_interwordspace_attr",1)}
66     %\lua_now:e{ltx.__tag.func.setinterwordspace()}
67     %\lua_now:e{ltx.__tag.func.setinterwordfont()}
68     %\lua_now:e{tex.setattribute("g__tag_interwordfont_attr",font.current())}
69     \lua_now:e{ltx.__tag.func.fakespace()}
70     \skip_horizontal:n{\c_zero_skip}
71     \group_end:
72   }
73 }
74 \end{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 | |
|--|---|
| <code>_</code> | 159, 172 |
| A | |
| <code>activate</code> | 150 |
| <code>activate-all</code> | 150 |
| <code>activate-mc</code> | 150 |
| <code>activate-struct</code> | 150 |
| <code>activate-tree</code> | 150 |
| <code>actualtext</code> | 38, 57, 269, 277, 522 |
| <code>actualtext-o</code> | 38, 57, 277, 522 |
| <code>actualtext-o_{uuuu}lang</code> | 269 |
| <code>add-new-tag</code> | 438 |
| <code>\AddToHook</code> | 13, 16, 51, 151, 165 |
| <code>AF</code> | 57, 396 |
| <code>AFinline</code> | 57, 396 |
| <code>AFinline-o</code> | 57, 396 |
| <code>alttext</code> | 38, 57, 269, 277, 522 |
| <code>alttext-o</code> | 38, 57, 269, 277, 522 |
| <code>artifact</code> | 38, 277, 522 |
| artifact-bool internal commands: | |
| <code>__artifact-bool</code> | 76 |
| artifact-type internal commands: | |
| <code>__artifact-type</code> | 76 |
| <code>attr-unknown</code> | 31 |
| <code>attribute</code> | 58, 628 |
| <code>attribute-class</code> | 58, 594 |
| B | |
| bool commands: | |
| <code>\bool_gset_false:N</code> | 31, 244, 502 |
| <code>\bool_gset_true:N</code> | 30, 215, 472 |
| <code>\bool_if:NTF</code> 9, 16, 23, 69, 118, 133, 154, 157, 157, 167, 170, 183, 212, 217, 224, 230, 231, 346, 479, 481, 489 | |
| <code>\bool_if:nTF</code> | 6 |
| <code>\bool_lazy_all:nTF</code> | 45 |
| <code>\bool_lazy_and:nnTF</code> | 62, 72 |
| <code>\bool_lazy_and_p:nn</code> | 8 |
| <code>\bool_new:N</code> | 14, 15, 29, 55, 80, 81, 82, 83, 85, 87, 112, 142, 143 |
| <code>\bool_set_false:N</code> | 146, 147, 179, 473, 503 |
| <code>\bool_set_true:N</code> | 84, 86, 178 |
| C | |
| <code>\c</code> | 134, 135 |
| c@g internal commands: | |
| <code>\c@g__tag_MCID_abs_int</code> | 9, 45, 109, 131, 136, 165, 233, 475 |
| <code>\c@g__tag_parenttree_obj_int</code> ... | 52 |
| <code>\c@g__tag_struct_abs_int</code> | 6, 46, 98, 101, 103, 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, 621, 624, 666 |
| clist commands: | |
| <code>\clist_const:Nn</code> 60, 77, 78, 106, 128, 144 | |
| <code>\clist_map_inline:Nn</code> 356, 361, 366, 373 | |
| <code>\clist_map_inline:nn</code> | 373 |
| <code>\clist_new:N</code> | 75 |
| <code>\clist_set:Nn</code> | 598, 632 |
| color commands: | |
| <code>\color_select:n</code> | 159, 172 |
| cs commands: | |
| <code>\cs_generate_variant:Nn</code> | 89, 90, 91, 92, 93, 94, 95, 96, 100, 112, 118, 123, 128, 128, 137, 138, 139, 140, 140, 141, 142, 182, 408, 410, 437, 465, 567 |
| <code>\cs_if_exist:NTF</code> | 53 |
| <code>\cs_if_exist_p:N</code> | 9 |
| <code>\cs_new:Nn</code> | 20, 68, 208, 386, 412, 434, 439, 443 |
| <code>\cs_new:Npn</code> | 9, 42, 55, 56, 61, 119, 124, 261, 521, 568 |
| <code>\cs_new_protected:Nn</code> | 62, 239, 388, 418, 466, 497, 504, 510, 533 |
| <code>\cs_new_protected:Npn</code> | 15, 24, 31, 32, 35, 37, 44, 56, 60, 66, 69, 80, 88, 89, 95, 101, 103, 105, 112, 113, 115, 118, 129, 130, 131, 131, 139, 141, 143, 146, 147, 154, 158, 170, 172, 177, 180, 183, 187, 188, 193, 197, 201, 209, 210, 210, 226, 227, 249, 444, 558, 581 |
| <code>\cs_set:Npn</code> | 38, 43 |
| <code>\cs_set_eq:NN</code> | 46, 47, 48, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 149 |
| <code>\cs_set_protected:Nn</code> | 9, 16, 23, 30, 49, 56 |
| <code>\cs_to_str:N</code> 12, 19, 26, 33, 52, 53, 59, 60 | |

| | |
|---|-----------------------|
| D | |
| \DeclareDocumentMetadata | 21 |
| \DeclareOption | 30, 31 |
| \documentclass | 22 |
| E | |
| E | 57, 269 |
| \ExecuteOptions | 32 |
| exp commands: | |
| \exp_args:Ne | 263, 453 |
| \exp_args:Nee | 57 |
| \exp_args:NNno | 459 |
| \exp_args:NNx | 39 |
| \exp_args:NNx | 39 |
| \exp_args:Nnx .. 56, 249, 331, 634, 638 | |
| \exp_args:NV | 206, 228, 485 |
| \exp_args:Nx | 216, 491 |
| \exp_not:n | 170 |
| F | |
| fi commands: | |
| \fi: | 19 |
| file commands: | |
| \file_input:n | 179 |
| \fontencoding | 6 |
| \fontfamily | 6 |
| \fontseries | 6 |
| \fontshape | 6 |
| \fontsize | 6 |
| G | |
| group commands: | |
| \group_begin: | |
| 64, 145, 213, 413, 429, 448, 470 | |
| \group_end: | |
| 71, 149, 236, 425, 441, 494, 499 | |
| H | |
| hook commands: | |
| \hook_gput_code:nnn ... 7, 21, 53, | |
| 180, 193, 203, 216, 222, 226, 344, 363 | |
| I | |
| if commands: | |
| \if_mode_horizontal: | 19 |
| \ignorespaces | 21, 13 |
| int commands: | |
| \int_case:nnTF | 143 |
| \int_compare:nNnTF | |
| 73, 148, 151, 194, 373, 513 | |
| \int_compare:nTF | |
| . 77, 203, 228, 614, 616, 618, 636, 662 | |
| \int_eval:n 88, 153, 170, 231, | |
| 280, 286, 299, 311, 323, 335, 347, | |
| 359, 366, 379, 391, 403, 450, 451, | |
| 452, 455, 458, 462, 483, 621, 624, 666 | |
| \int_gincr:N | 131, 153, 449, 475 |
| \int_gset:Nn | 55, 149 |
| \int_gzero:N | 8, 157 |
| \int_new:N | 10, 76, 79, 144 |
| \int_rand:n .. 39, 40, 42, 44, 46, 48, 49 | |
| \int_set:Nn ... 158, 159, 160, 161, 162 | |
| \int_step_inline:nnnn | |
| 46, 71, 74, 91, 188, 194, 379 | |
| \int_to_Hex:n 39, 40, 42, 44, 46, 48, 49 | |
| \int_use:N | 9, 44, 45, 98, |
| 101, 103, 107, 109, 111, 126, 136, | |
| 159, 165, 172, 233, 414, 418, 419, | |
| 422, 424, 430, 434, 435, 438, 440, 568 | |
| interwordspace | 6 |
| iow commands: | |
| \iow_newline: | 171 |
| \iow_now:Nn | 39 |
| K | |
| keys commands: | |
| \keys_define:nn | 12, 20, 32, |
| 51, 54, 66, 76, 128, 145, 150, 269, | |
| 277, 396, 438, 446, 522, 587, 594, 628 | |
| \keys_set:nn | |
| ... 9, 51, 216, 332, 461, 476, 635, 639 | |
| \keys_set_known:nnnN | 450 |
| L | |
| label | 38, 57, 269, 277, 522 |
| lang | 57 |
| legacy commands: | |
| \legacy_if:nTF | 37 |
| \llap | 159 |
| log | 157 |
| lua commands: | |
| \lua_now:n | 8, 12, |
| 19, 26, 33, 36, 39, 40, 42, 44, 45, 52, | |
| 53, 59, 60, 60, 65, 66, 67, 68, 69, 77, | |
| 85, 97, 122, 348, 356, 365, 376, 389, | |
| 390, 399, 414, 423, 436, 441, 452, | |
| 528, 536, 550, 567, 584, 601, 614, 624 | |
| M | |
| mc-current | 22, 14, 66 |
| mc-data | 22, 54 |
| mc-label-unknown | 9 |
| mc-nested | 6 |
| mc-not-open | 11 |
| mc-popped | 12 |
| mc-pushed | 12 |
| mc-tag-missing | 8 |
| mc-used-twice | 10 |

\MessageBreak 15, 19, 20, 21
msg commands:
 \msg_error:nn 85, 103, 237, 473
 \msg_error:nnn 162, 224, 604, 642
 \msg_info:nnn .. 97, 150, 230, 392, 420
 \msg_info:nnnn 124
 \g_msg_module_name_prop 25, 27
 \msg_new:nnn 7, 8,
 9, 10, 11, 12, 13, 14, 20, 21, 24, 25,
 27, 29, 31, 32, 33, 34, 35, 36, 37, 39, 40
 \msg_note:nn 124
 \msg_redirect_name:nnn 17, 35
 \msg_warning:nn 108
 \msg_warning:nnn
 23, 54, 56, 92, 112, 119, 127,
 135, 143, 154, 166, 174, 256, 515, 554

N

new-tag 35
newattribute 58, 581
\newcommand 178, 179
\newcounter 7, 8, 52
\NewDocumentCommand 7,
11, 17, 23, 28, 32, 37, 42, 49, 137
\newlabeldata 41

O

obj-write-num 39

P

\PackageError 13
paratagging 23, 145
paratagging-show 23, 145
pdf commands:
 \pdf_bdc:nn 127
 \pdf_bmc:n 125
 \pdf_emc: 126
 \pdf_name_from_unicode_e:n
 282, 405, 431
 \pdf_object_if_exist:n 88
 \pdf_object_if_exist:nTF
 100, 101, 212, 400, 414, 430
 \pdf_object_new:nn
 18, 20, 20, 51, 146, 176, 186, 454
 \pdf_object_ref:n 29, 34,
 37, 41, 88, 89, 102, 103, 108, 183,
 198, 255, 405, 424, 433, 440, 486, 550
 \pdf_object_ref_last: 124, 651
 \pdf_object_unnamed_write:nn 116, 646
 \pdf_object_write:nn
 141, 149, 177, 193, 200, 205, 217
 \pdf_pageobject_ref:n 98
 \pdf_string_from_unicode:nnN ... 26
 \pdf_uncompress: 175

 \pdf_version_compare:NnTF
 51, 371, 410, 479
pdfannot commands:
 \pdfannot_dict_put:nnn
 90, 187, 210, 228, 233
 \pdfannot_link_ref_last: ... 197, 220
pdffdict commands:
 \pdffdict_gput:nnn
 22, 27, 32, 197, 403, 428
 \pdffdict_if_empty:nTF 191
 \pdffdict_new:n 19, 21, 387
 \pdffdict_use:n 151, 195, 202
\pdffakespace 22, 135
pdffile commands:
 \pdffile_embed_stream:nnn 91, 416, 432
\pdfglyphtounicode 11
\pdfinterwordspacelon 14, 15
pdfmanagement commands:
 \pdfmanagement_add:nnn
 25, 26, 167, 169, 171, 228
 \pdfmanagement_if_active_p: .. 9, 10
 \pdfmanagement_remove:nn 173
prg commands:
 \prg_generate_conditional_-
 variant:Nnn 88
 \prg_new_conditional:Nnn ... 116, 371
 \prg_new_conditional:Npnn . 43, 60, 70
 \prg_new_eq_conditional:NNn 123, 385
 \prg_return_false: 57, 67, 77, 120, 381
 \prg_return_true: . 54, 64, 74, 119, 382
\ProcessOptions 33
prop commands:
 \prop_clear:N 73
 \prop_const_from_keyval:Nn 339
 \prop_count:N 94
 \prop_get:NnNTF
 96, 107, 122, 141, 283, 465
 \prop_gput:Nnn
 25, 27, 34, 92, 130, 189, 259, 359,
 364, 369, 376, 398, 426, 459, 583, 651
 \prop_if_exist:NnTF 25, 537
 \prop_if_in:NnTF 58,
 82, 90, 164, 201, 390, 470, 602, 640, 644
 \prop_item:Nn 30, 62, 83,
 133, 162, 205, 266, 275, 354, 649, 656
 \prop_map_inline:Nn 189, 412
 \prop_map_tokens:Nn 207
 \prop_new:N 9, 10, 11, 72, 128, 577, 580
 \prop_put:Nnn 80, 93
 \prop_show:N 58, 135, 494, 497, 624, 645
\ProvidesExplFile 3
\ProvidesExplPackage
 ... 3, 3, 3, 3, 3, 3, 3, 108, 340, 573

| | |
|--------------------------------|--|
| R | |
| raw | 38 , 277 , 522 |
| ref | 57 , 269 |
| ref commands: | |
| \ref_attribute_gset:nnnn | 97 , 99 , 106 , 108 , 110 |
| \ref_label:nn | 94 , 115 |
| \ref_value:nn | 376 |
| \ref_value:nnn | 6 , 53 , 53 , 55 , 121 , 126 |
| ref internal commands: | |
| __ref_value:nnn | 58 , 61 |
| regex commands: | |
| \regex_replace_once:nnN | 133 |
| \RequirePackage | 20 , 34 , 185 , 188 |
| \rlap | 172 |
| role | 438 |
| role-missing | 32 |
| role-namespace | 438 |
| role-tag | 35 |
| role-unknown | 32 |
| role-unknown-tag | 32 |
| S | |
| \selectfont | 6 |
| seq commands: | |
| \seq_clear:N | 193 |
| \seq_const_from_clist:Nn | 16 , 28 |
| \seq_count:N | 145 , 379 , 396 , 424 , 614 , 616 , 618 , 636 , 662 |
| \seq_get:NNTF | 233 , 469 , 520 , 527 |
| \seq_gpop:NN | 510 |
| \seq_gpop:NNTF | 60 , 511 |
| \seq_gpop_left:NN | 132 |
| \seq_gpush:Nn | 11 , 13 , 43 , 50 , 475 , 476 |
| \seq_gput_left:Nn | 137 , 606 |
| \seq_gput_right:Nn | 32 , 131 , 177 , 213 , 446 |
| \seq_gremove_duplicates:N | 157 |
| \seq_if_in:NnTF | 172 |
| \seq_item:Nn | 132 , 164 , 276 , 277 , 353 , 383 , 460 , 461 |
| \seq_log:N | 131 , 155 |
| \seq_map_inline:Nn | 191 , 600 , 638 |
| \seq_new:N | 10 , 12 , 12 , 18 , 73 , 74 , 129 , 169 , 578 |
| \seq_set_from_clist:NN | 599 , 633 |
| \seq_set_map:NNn | 158 , 608 |
| \seq_set_split:Nnn | 95 , 275 , 459 |
| \seq_show:N | 51 , 132 , 134 , 223 , 478 , 495 , 498 , 507 |
| \seq_use:Nn | 169 , 176 , 615 |
| \l_tmpa_seq | 193 , 213 , 223 , 459 , 460 , 461 |
| shipout commands: | |
| \g_shipout_readonly_int | 44 , 107 , 126 , 231 |
| show-spaces | 6 |
| \ShowTagging | 22 , 49 |
| \showtagpdfmcddata | 12 |
| skip commands: | |
| \skip_horizontal:n | 70 |
| \c_zero_skip | 70 |
| stash | 38 , 57 , 76 , 269 |
| \stepcounter | 258 |
| str commands: | |
| \str_const:Nn | 37 |
| \str_new:N | 71 |
| \str_set_convert:Nnnn | 96 , 290 , 293 , 300 , 305 , 310 , 317 , 320 , 329 , 341 , 353 , 385 , 543 , 560 , 577 , 594 |
| \str_use:N | 554 , 571 , 588 , 607 |
| \l_tmpa_str | 26 , 30 |
| \string | 20 , 21 , 22 |
| struct-faulty-nesting | 21 |
| struct-label-unknown | 27 |
| struct-missing-tag | 24 |
| struct-no-objnum | 20 |
| struct-show-closing | 29 |
| struct-stack | 22 , 128 |
| struct-used-twice | 25 |
| sys commands: | |
| \sys_if_engine luatex:TF | 30 , 30 , 46 , 47 , 58 , 60 , 71 , 135 , 177 |
| \sys_if_engine pdftex:TF | 7 , 48 |
| \sys_if_engine xetex:TF | 49 |
| \sys_if_output_pdf:TF | 9 , 11 |
| sys-no-interwordspace | 40 |
| T | |
| tabsorder | 165 |
| tag | 38 , 56 , 269 , 277 , 438 , 522 |
| tag commands: | |
| \tag_get:n | 12 , 66 , 42 , 42 , 43 , 46 |
| \tag_if_active: | 43 |
| \tag_if_active:TF | 12 , 43 |
| \tag_if_active_p: | 12 , 43 |
| \tag_mc_artifact_group_begin:n | 37 , 24 , 24 |
| \tag_mc_artifact_group_end: | 37 , 24 , 31 |
| \tag_mc_begin:n | 7 , 37 , 13 , 27 , 68 , 158 , 162 , 171 , 186 , 209 , 209 , 466 , 466 |
| \tag_mc_begin_pop:n | 37 , 35 , 37 , 56 , 200 , 223 |
| \tag_mc_end: | 37 , 20 , 34 , 47 , 160 , 169 , 173 , 198 , 209 , 221 , 239 , 497 , 497 |

| | |
|---|--|
| \tag_mc_end_push: | 37, 26, 37, 37, 184, 207 |
| \tag_mc_if_in: | 123, 371, 385 |
| \tag_mc_if_in:TF | 37, 30, 116 |
| \tag_mc_if_in_p: | 37, 116 |
| \tag_mc_use:n 37, 25, 249, 249, 497, 510 | |
| \tag_stop_group_begin: .. | 28, 143, 143 |
| \tag_stop_group_end: ... | 33, 143, 149 |
| \tag_struct_begin:n | 56, 34, 156, 185, 208, 444, 444 |
| \tag_struct_end: | 56, 39, 175, 199, 222, 444, 504 |
| \tag_struct_insert_annot:nn | 56, 72, 197, 220, 558, 558, 567 |
| \tag_struct_parent_int: | 56, 72, 190, 197, 213, 220, 558, 568 |
| \tag_struct_use:n ... | 56, 44, 533, 533 |
| tag internal commands: | |
| \g__tag_active_mc_bool | 48, 62, 80, 152, 346 |
| \l__tag_active_mc_bool | 51, 62, 83, 147 |
| \g__tag_active_struct_bool | 47, 72, 80, 154, 230 |
| \l__tag_active_struct_bool | 50, 72, 83, 146 |
| \g__tag_active_tree_bool | 9, 23, 49, 80, 153, 212, 224 |
| \g__tag_attr_class_used_seq | 157, 158, 576, 606 |
| \g__tag_attr_entries_prop | 163, 576, 583, 602, 640, 645, 649 |
| __tag_attr_new_entry:nn | 581, 581, 591 |
| \g__tag_attr_objref_prop | 576, 644, 651, 656 |
| \l__tag_attr_value_tl | 576, 634, 653, 658, 660, 664, 668 |
| __tag_check_add_tag_role:nn ... | 115, 115, 400, 427 |
| __tag_check_if_active_mc: | 60 |
| __tag_check_if_active_mc:TF ... | 39, 58, 60, 211, 241, 468, 499 |
| __tag_check_if_active_struct: .. | 70 |
| __tag_check_if_active_struct:TF | 60, 251, 446, 508, 535, 561 |
| __tag_check_info_closing_-struct:n | 95, 95, 100, 515 |
| __tag_check_mc_if_nested: | 131, 131, 214, 471 |
| __tag_check_mc_if_open: | 131, 139, 243, 501 |
| __tag_check_mc_pushed_popped:nn | 44, 51, 64, 67, 72, 146, 146 |
| __tag_check_mc_tag:N | 158, 158, 222, 478 |
| __tag_check_mc_used:n | 170, 170, 185, 445 |
| \g__tag_check_mc_used_seq | 169, 172, 177 |
| __tag_check_no_open_struct: ... | 101, 101, 518, 525 |
| __tag_check_record_pdfobj_num:n | 226, 226 |
| __tag_check_show_MCID_by_page: . | 180, 180 |
| __tag_check_struct_used:n | 105, 105, 540 |
| __tag_check_structure_has_tag:n | 80, 80, 462 |
| __tag_check_structure_tag:N ... | 88, 88, 278 |
| __tag_fakespace: | 60, 62, 139 |
| __tag_finish_structure: | 13, 16, 210, 210 |
| __tag_get_data_mc_tag: | 208, 208, 521, 521 |
| __tag_get_data_struct_tag: 261, 261 | |
| __tag_get_mc_abs_cnt: 9, 9, 17, 18, | 60, 90, 101, 135, 143, 162, 405, 491, |
| | 530, 538, 554, 571, 588, 605, 618, 628 |
| \g__tag_in_mc_bool | 16, 111, 118, 215, 244, 472, 502 |
| __tag_lastpagelabel: ... | 35, 35, 52 |
| \l__tag_loglevel_int ... | 79, 149, |
| | 152, 158, 159, 160, 161, 162, 228, 513 |
| \l__tag_mc_artifact_bool | 14, 79, 217, 473, 481, 503 |
| \l__tag_mc_artifact_type_tl ... | 13, 83, 87, 91, 95, 99, 103, 219, 641 |
| __tag_mc_bdc:nn 124, 127, 128, 168, 199 | |
| __tag_mc_bdc_mcid:n | 129, 172 |
| __tag_mc_bdc_mcid:nn | 129, 129, 174, 179 |
| __tag_mc_bmc:n | 124, 125, 195 |
| __tag_mc_bmc_artifact: 193, 193, 205 | |
| __tag_mc_bmc_artifact:n 193, 197, 206 | |
| __tag_mc_emc: | 124, 126, 246 |
| __tag_mc_handle_artifact:N | 193, 201, 219 |
| __tag_mc_handle_mc_label:n ... | 20, 20, 229, 486 |
| __tag_mc_handle_mcid:nn | 129, 177, 182, 223 |
| __tag_mc_handle_stash:n | 183, 183, 233, 443, 443, 465, 491, 518 |
| __tag_mc_if_in: 116, 123, 371, 371, 385 | |
| __tag_mc_if_in:TF .. | 41, 116, 133, 141 |
| __tag_mc_if_in_p: | 116 |

```

\__tag_mc_insert_mcid_kids:n ...
..... 134, 434, 434, 450
\__tag_mc_insert_mcid_single_-
kids:n ..... 135, 434, 439
\l__tag_mc_key_label_tl .....
..... 16, 226, 229, 328, 483, 486, 613
\l__tag_mc_key_properties_tl ...
..... 16, 225, 286, 295, 296, 305,
306, 315, 316, 325, 326, 474, 535,
548, 549, 565, 566, 582, 583, 599, 600
\l__tag_mc_key_stash_bool .....
..... 14, 78, 231, 489
\g__tag_mc_key_tag_tl .....
.. 16, 17, 208, 245, 282, 506, 521, 527
\l__tag_mc_key_tag_tl .....
. 16, 222, 224, 281, 478, 480, 505, 526
\__tag_mc_lua_set_mc_type_attr:n
..... 386, 386, 410, 480
\__tag_mc_lua_unset_mc_type_-
attr: ..... 386, 412, 504
\g__tag_mc_parenttree_prop .....
..... 11, 12, 83, 189, 260, 460
\l__tag_mc_ref_abspage_tl .....
..... 114, 132, 144, 152, 160
\g__tag_mc_stack_seq 12, 43, 50, 60, 155
\l__tag_mc_tmpa_tl . 115, 146, 149, 153
g__tag_MCID_abs_int ..... 7
\g__tag_MCID_byabspage_prop ....
..... 113, 142, 151, 159
\g__tag_MCID_tmp_bypage_int ....
..... 10, 111, 149, 157, 170
\g__tag_mode_lua_bool .....
..... 29, 30, 31, 69, 133, 157, 183
\__tag_new_output_prop_handler:n
..... 56, 66, 75, 451
\l__tag_para_bool .....
..... 142, 147, 154, 167, 178, 179
\g__tag_para_int ... 142, 153, 159, 172
\l__tag_para_show_bool .....
..... 142, 148, 157, 170
\__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_prop_gput:Nnn .....
..... 9, 23, 34, 38, 78, 85, 128,
130, 137, 150, 158, 162, 173, 180,
279, 285, 298, 310, 322, 334, 346,
358, 365, 378, 381, 390, 394, 402,
421, 422, 437, 457, 482, 546, 620, 665
\__tag_prop_item:Nn .. 9, 43, 128, 133
\__tag_prop_new:N ..... 8,
9, 9, 11, 74, 113, 128, 128, 139, 450

\__tag_prop_show:N 9, 56, 128, 135, 142
\__tag_ref_label:nn .....
..... 22, 112, 112, 118, 163, 466
\__tag_ref_value:nnn .....
..... 78, 82, 98, 99, 108, 119, 119, 123,
134, 205, 216, 253, 512, 538, 544, 547
\__tag_ref_value_lastpage:nn ...
..... 57, 71, 74, 124, 124, 184, 198
\c__tag_refmc_clist ..... 77
\c__tag_refstruct_clist ..... 77
g__tag_role/RoleMap_dict ..... 387
\__tag_role_add_tag:nn .....
..... 388, 388, 408, 414, 482
\__tag_role_add_tag:nnnn .....
..... 418, 418, 437, 487
\__tag_role_NS_new:nnn .....
..... 92, 15, 15, 53, 54, 55, 57
\g__tag_role_NS_prop .....
..... 10, 34, 189, 207, 283, 470
\l__tag_role_role_namespace_-
tmpa_tl ..... 11,
443, 463, 468, 470, 472, 476, 491
\l__tag_role_role_tmpa_tl .....
..... 11, 442, 461, 467, 484, 490
\c__tag_role_sttags_mathml_clist
..... 59, 373
\c__tag_role_sttags_only_pdf_-
clist ..... 59, 361
\c__tag_role_sttags_only_pdfII_-
clist ..... 59, 366
\c__tag_role_sttags_pdf_pdfII_-
clist ..... 59, 356
\c__tag_role_sttags_pdfII_to_-
pdf_prop ..... 59, 412
\l__tag_role_tag_namespace_tmpa_-
tl ..... 11, 441, 489
\l__tag_role_tag_tmpa_tl .....
..... 11, 440, 460, 483, 488
\g__tag_role_tags_NS_prop 9, 164,
275, 359, 364, 369, 376, 398, 426, 466
\g__tag_role_tags_prop .....
... 6, 90, 122, 354, 381, 390, 394, 422
\g__tag_role_tags_seq .....
..... 6, 353, 358, 363,
368, 375, 379, 383, 393, 396, 421, 424
\c__tag_role_userNS_id_str 91, 37, 57
\__tag_seq_gput_right:Nn .....
..... 9, 30, 91, 105, 121, 128,
131, 138, 358, 363, 368, 375, 393, 421
\__tag_seq_item:Nn ... 9, 38, 128, 132
\__tag_seq_new:N .....
... 7, 9, 9, 16, 76, 128, 129, 140, 452
\__tag_seq_show:N . 9, 49, 128, 134, 141
\l__tag_showspaces_bool ... 16, 24, 57

```

| | | |
|--|--|--|
| <code>g__tag_struct_0_prop</code> | 74 | 325, 326, 330, 337, 342, 349, 354, |
| <code>\l__tag_struct_elem_stash_bool</code> .. | | 361, 386, 393, 544, 549, 554, 561, |
| | 55, 272, 480 | 566, 571, 578, 583, 588, 595, 600, 607 |
| <code>__tag_struct_exchange_kid_-</code> | | <code>\l__tag_tmpa_tl</code> 60, 62, 67, 68, 70, |
| <code>command:N</code> | 130, 130, 140, 159 | 96, 97, 100, 102, 132, 136, 137, 156, |
| <code>__tag_struct_fill_kid_key:n</code> ... | | 167, 174, 179, 182, 190, 215, 220, |
| | 141, 141, 214 | 253, 254, 262, 272, 283, 288, 372, |
| <code>__tag_struct_get_dict_content:nN</code> | | 375, 381, 389, 396, 510, 511, 512, |
| | 188, 188, 215 | 513, 518, 520, 522, 527, 529, 612, 623 |
| <code>__tag_struct_insert_annot:nn</code> ... | | <code>\l__tag_tmpb_seq</code> |
| | 227, 227, 563 | 70, 608, 615 |
| <code>\l__tag_struct_key_label_tl</code> | | <code>__tag_tree_fill_parenttree:</code> ... |
| | 54, 271, 464, 466 | |
| <code>__tag_struct_kid_mc_gput_-</code> | | <code>__tag_tree_lua_fill_parenttree:</code> |
| <code>right:nn</code> | 89, 89, 186, 267 | |
| <code>__tag_struct_kid_OBJR_gput_-</code> | | <code>__tag_tree_write_classmap:</code> |
| <code>right:nn</code> | 113, 113, 128, 241 | |
| <code>__tag_struct_kid_struct_gput_-</code> | | <code>__tag_tree_write_namespaces:</code> ... |
| <code>right:nn</code> ... | 103, 103, 112, 491, 542 | |
| <code>g__tag_struct_kids_0_seq</code> | 74 | 187, 187, 217 |
| <code>\g__tag_struct_objR_seq</code> | 9 | <code>__tag_tree_write_parenttree:</code> ... |
| <code>__tag_struct_output_prop_aux:nn</code> | | |
| | 56, 56, 70 | <code>__tag_tree_write_rolemap:</code> |
| <code>\g__tag_struct_stack_current_tl</code> . | | |
| | 14, 73, 187, 191, | <code>__tag_tree_write_structelements:</code> |
| | 265, 266, 269, 448, 456, 462, 477, | |
| | 489, 493, 494, 497, 515, 522, 543, 550 | <code>__tag_tree_write_structtreeroot:</code> |
| <code>\l__tag_struct_stack_parent_-</code> | | |
| <code>tmpa_tl</code> | 14, 235, | <code>tag-namespace</code> |
| | 243, 255, 471, 486, 490, 492, 495, 498 | 438 |
| <code>\g__tag_struct_stack_seq</code> | | <code>tag/struct/0 internal commands:</code> |
| . | 10, 234, 470, 475, 478, 507, 511, 520 | <code>__tag/struct/0</code> |
| <code>\c__tag_struct_StructElem_-</code> | | 20 |
| <code>entries_seq</code> | 16 | <code>tag/tree/namespaces internal commands:</code> |
| <code>\c__tag_struct_StructTreeRoot_-</code> | | <code>__tag/tree/namespaces</code> |
| <code>entries_seq</code> | 16 | 186 |
| <code>\g__tag_struct_tag_NS_tl</code> 52, 277, 283 | | <code>tag/tree/parenttree internal commands:</code> |
| <code>\g__tag_struct_tag_stack_seq</code> ... | | <code>__tag/tree/parenttree</code> |
| | 12, 131, 132, 476, 510, 527 | 51 |
| <code>\g__tag_struct_tag_tl</code> | | <code>tag/tree/rolemap internal commands:</code> |
| | 52, 276, 278, 282, 476, 529 | <code>__tag/tree/rolemap</code> |
| <code>__tag_struct_write_obj:n</code> | | 146 |
| | 42, 48, 210, 210 | <code>tagabspage</code> |
| <code>\g__tag_tagunmarked_bool</code> | 87, 163 | 97 |
| <code>\l__tag_tmpa_clist</code> | | <code>tagmcabs</code> |
| | 70, 598, 599, 632, 633 | 97 |
| <code>\l__tag_tmpa_int</code> | 70 | <code>\tagmcbegin</code> |
| <code>\l__tag_tmpa_prop</code> .. | 70, 73, 81, 94, 96 | 21, 11 |
| <code>\l__tag_tmpa_seq</code> | 70, | <code>\tagmcend</code> |
| | 158, 170, 275, 276, 277, 599, 600, | 21, 11 |
| | 608, 614, 616, 618, 633, 636, 638, 662 | <code>tagmcid</code> |
| <code>\l__tag_tmpa_str</code> | | 97 |
| | 70, 291, 294, 296, 301, 301, | <code>\tagmcifin</code> |
| | 306, 306, 311, 313, 316, 318, 321, | 21 |
| | | <code>\tagmcifinTF</code> |
| | | 28 |
| | | <code>\tagmcuse</code> |
| | | 21, 11 |
| | | <code>\tagpdfifluatexT</code> |
| | | 46 |
| | | <code>\tagpdfifluatexTF</code> |
| | | 46 |
| | | <code>\tagpdfifpdfTeX</code> |
| | | 48 |
| | | <code>\tagpdfifpdfTeXTF</code> |
| | | 46 |
| | | <code>\tagpdfparaOff</code> |
| | | 23, 178 |
| | | <code>\tagpdfparaOn</code> |
| | | 23, 178 |
| | | <code>\tagpdfsetup</code> |
| | | 21, 58, 6 |
| | | <code>tagstruct</code> |
| | | 97 |
| | | <code>\tagstructbegin</code> |
| | | 21, 32 |
| | | <code>\tagstructend</code> |
| | | 21, 32 |
| | | <code>tagstructobj</code> |
| | | 97 |

| | | | |
|---|---|--|--|
| <code>\tagstructuse</code> | 21 , 32 | <code>\tl_new:N</code> | 11 , |
| <code>tagunmarked</code> | 163 | | 12 , 13 , 13 , 14 , 14 , 15 , 16 , 17 , 18 , |
| <code>T_EX</code> and <code>L^AT_EX 2_ε</code> commands: | | | 19 , 52 , 53 , 54 , 59 , 67 , 70 , 114 , 115 , 579 |
| <code>\@auxout</code> | 39 | <code>\tl_put_right:Nn</code> . | 86 , 98 , 111 , 140 , |
| <code>\@bsphack</code> | 114 | | 198 , 286 , 295 , 296 , 305 , 306 , 315 , |
| <code>\@esphack</code> | 116 | | 316 , 325 , 326 , 375 , 535 , 548 , 549 , |
| <code>\@gobble</code> | 361 | | 565 , 566 , 582 , 583 , 599 , 600 , 653 , 660 |
| <code>\@secondoftwo</code> | 361 | <code>\tl_set:Nn</code> | |
| <code>\tiny</code> | 159 , 172 | | 83 , 87 , 91 , 95 , 99 , 103 , 120 , 132 , |
| <code>title</code> | 57 , 269 | | 167 , 182 , 253 , 281 , 389 , 460 , 461 , |
| <code>title-o</code> | 57 , 269 | | 472 , 476 , 505 , 512 , 526 , 612 , 613 , 634 |
| <code>tl</code> commands: | | <code>\tl_show:N</code> | 489 , 490 , 658 , 664 |
| <code>\c_space_tl</code> 62 , 64 , 88 , 89 , 95 , 97 , 99 , | | <code>\tl_tail:n</code> | 264 |
| 104 , 143 , 160 , 181 , 205 , 432 , 615 , 655 | | <code>\tl_to_str:n</code> | 59 |
| <code>\tl_clear:N</code> | 156 , 190 , 372 , 474 | <code>\tl_use:N</code> | 64 |
| <code>\tl_gput_right:Nn</code> | 62 | <code>\l_tmpa_tl</code> | 110 , 122 , 456 , 457 , 459 |
| <code>\tl_gset:Nn</code> | 73 , 245 , | token commands: | |
| 276 , 277 , 282 , 477 , 506 , 522 , 527 , 529 | | <code>\token_to_str:N</code> | 41 |
| <code>\tl_if_empty:NnTF</code> | 160 , 173 , | <code>tree-mcid-index-wrong</code> | 37 |
| 204 , 226 , 254 , 457 , 463 , 463 , 483 , 513 | | | |
| <code>\tl_if_empty:nTF</code> | 117 , 401 | | |
| <code>\tl_if_eq:NnTF</code> | 62 | | |
| <code>\tl_if_exist:NnTF</code> | 63 | | |

U

| | |
|-------------------------------|---|
| <code>\unskip</code> | 21 , 19 |
| use commands: | |
| <code>\use:N</code> | 42 |
| <code>\use_ii:nn</code> | 207 |