

The **tagpdf-mc** module

Code related to Marked Content (mc-chunks)

part of the tagpdf package

Ulrike Fischer*

Version 0.82, released 2021-06-14

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.

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.

*E-mail: fischer@troubleshooting-tex.de

<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`.

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

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

<code>raw</code>	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.
------------------	---

<code>alttext</code>	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.
<code>alttext-o</code>	

<code>actualtext</code>	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.
<code>actualtext-o</code>	

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

<code>stash</code>	This “stashes” an mc-chunk: it is not inserted into the current structure. It should be normally be used along with a label to be able to use the mc-chunk in another place. The code is splitted into three parts: code shared by all engines, code specific to <code>luamode</code> and code not used by <code>luamode</code> .
--------------------	--

3 Marked content code – shared

```

1 <@@=tag>
2 <*shared>
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 }

```

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`

```
6 \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.

```
7 \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.

```
8 \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

```
9 \__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:

```
10 \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.

```
11 \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`

```
12 \bool_new:N \l__tag_mc_key_stash_bool
```

```
13 \bool_new:N \l__tag_mc_artifact_bool
```

(End definition for \l__tag_mc_key_stash_bool and \l__tag_mc_artifact_bool.)

`\l__tag_mc_key_tag_tl` Variables used by the keys. `\l_@@mc_key_properties_tl` will collect a number of values. TODO: should this be a pdfdict now?
`\g__tag_mc_key_tag_tl`
`\l__tag_mc_key_label_tl` 14 `\tl_new:N \l__tag_mc_key_tag_tl`
`\l__tag_mc_key_properties_tl` 15 `\tl_new:N \g__tag_mc_key_tag_tl`
16 `\tl_new:N \l__tag_mc_key_label_tl`
17 `\tl_new:N \l__tag_mc_key_properties_tl`
(End definition for `\l__tag_mc_key_tag_tl` and others.)

3.2 Functions

`__tag_mc_handle_mc_label:n` The commands labels a mc-chunk. It is used if the user explicitly labels the mc-chunk with the `label` key. The argument is the value provided by the user. It stores the attributes
`tagabspace`: the absolute page, `\g_shipout_readonly_int`,
`tagmcabs`: the absolute mc-counter `\c@g_@@MCID_abs_int`,
`tagmcid`: the ID of the chunk on the page `\g_@@MCID_tmp_bypage_int`, this typically settles down after a second compilation. The reference command is defined in `tagpdf.dtx` and is based on `l3ref`.
18 `\cs_new:Nn __tag_mc_handle_mc_label:n`
19 `{`
20 `__tag_ref_label:en{tagpdf-#1}{mc}`
21 `}`
(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
`\tag_mc_artifact_group_end:` creates a group.
22 `\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1`
23 `{`
24 `\tag_mc_begin:n {artifact=#1}`
25 `\tag_stop_group_begin:`
26 `}`
27
28 `\cs_new_protected:Npn \tag_mc_artifact_group_end:`
29 `{`
30 `\tag_stop_group_end:`
31 `\tag_mc_end:`
32 `}`
(End definition for `\tag_mc_artifact_group_begin:n` and `\tag_mc_artifact_group_end:`. These functions are documented on page 1.)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n` 33 `\cs_new_protected:Npn \tag_mc_end_push:`
34 `{`
35 `__tag_mc_if_in:TF`
36 `{`
37 `\seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }`
38 `__tag_check_mc_pushed_popped:nn`
39 `{ pushed }`
40 `{ \tag_get:n {mc_tag} }`
41 `\tag_mc_end:`

```

42     }
43     {
44         \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
45         \__tag_check_mc_pushed_popped:nn { pushed }{-1}
46     }
47 }
48
49 \cs_new_protected:Npn \tag_mc_begin_pop:n #1
50 {
51     \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
52     {
53         \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
54         {
55             \__tag_check_mc_pushed_popped:nn {popped}{-1}
56         }
57         {
58             \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
59             \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
60         }
61     }
62     {
63         \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
64     }
65 }

```

(End definition for `\tag_mc_end_push:` and `\tag_mc_begin_pop:n`. These functions are documented on page 1.)

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 66 \keys_define:nn { __tag / mc }
__artifact-type 67 {
68     stash .bool_set:N = \l__tag_mc_key_stash_bool,
69     __artifact-bool .bool_set:N = \l__tag_mc_artifact_bool,
70     __artifact-type .choice:,
71     __artifact-type / pagination .code:n =
72     {
73         \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
74     },
75     __artifact-type / layout .code:n =
76     {
77         \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
78     },
79     __artifact-type / page .code:n =
80     {
81         \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
82     },
83     __artifact-type / background .code:n =
84     {
85         \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
86     },

```

```

87   __artifact-type / notype      .code:n    =
88   {
89     \tl_set:Nn \l__tag_mc_artifact_type_tl {}
90   },
91   __artifact-type /      .code:n    =
92   {
93     \tl_set:Nn \l__tag_mc_artifact_type_tl {}
94   },
95 }

```

(End definition for `stash`, `__artifact-bool`, and `__artifact-type`. This function is documented on page 2.)

```

96 </shared>

```

4 Marked content code – generic mode

```

97 <*generic>
98 \ProvidesExplPackage {tagpdf-mc-code-generic} {2021-06-14} {0.82}
99 {part of tagpdf - code related to marking chunks - generic mode}

```

4.1 Variables

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

```

100 \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 $\langle abspagenum \rangle = \langle max\ mcid \rangle$

```

101 \__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.

```

102 \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

```

103 \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:TF
  \tag_mc_if_in_p: 104 \prg_new_conditional:Nnn \_tag_mc_if_in: {p,T,F,TF}
  \tag_mc_if_in:TF 105 {
                    106   \bool_if:NTF \g__tag_in_mc_bool
                    107     { \prg_return_true: }
                    108     { \prg_return_false: }
                    109   }
                    110
                    111 \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 2.)

`_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`

```
112 % #1 tag, #2 properties
113 \cs_set_eq:NN \_tag_mc_bmc:n \pdf_bmc:n
114 \cs_set_eq:NN \_tag_mc_emc: \pdf_emc:
115 \cs_set_eq:NN \_tag_mc_bdc:nn \pdf_bdc:nn
116 \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.
`_tag_mc_bdc_mcid:n`
`_tag_mc_handle_mcid:nn`
`_tag_mc_handle_mcid:VV`

```
117 \cs_new_protected:Npn \_tag_mc_bdc_mcid:nn #1 #2
118 {
119   \int_gincr:N \c@g__tag_MCID_abs_int
120   \tl_set:Nx \l__tag_mc_ref_abspage_tl
121   {
122     \_tag_ref_value:enn %3 args
123     {
124       mcid-\int_use:N \c@g__tag_MCID_abs_int
125     }
126     { tagabspace }
127     {-1}
128   }
129   \prop_get:NoNTF
130   \g__tag_MCID_byabspace_prop
131   {
132     \l__tag_mc_ref_abspage_tl
133   }
134   \l__tag_mc_tmpa_tl
135   {
136     %key already present, use value for MCID and add 1 for the next
137     \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
```

```

138     \__tag_prop_gput:Nxx
139     \g__tag_MCID_byabspage_prop
140     { \l__tag_mc_ref_abspage_tl }
141     { \int_eval:n { \l__tag_mc_tmpa_tl +1 } }
142   }
143   {
144     %key not present, set MCID to 0 and insert 1
145     \int_gzero:N \g__tag_MCID_tmp_bypage_int
146     \__tag_prop_gput:Nxx
147     \g__tag_MCID_byabspage_prop
148     { \l__tag_mc_ref_abspage_tl }
149     {1}
150   }
151   \__tag_ref_label:en
152   {
153     mcid-\int_use:N \c@g__tag_MCID_abs_int
154   }
155   { mc }
156   \__tag_mc_bdc:nx
157   {#1}
158   { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
159 }
160 \cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
161 {
162   \__tag_mc_bdc_mcid:nn {#1} {}
163 }
164
165 \cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 % #1 tag, #2 properties
166 {
167   \__tag_mc_bdc_mcid:nn {#1} {#2}
168 }
169
170 \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,

```

171 \cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
172 {
173   \__tag_check_mc_used:n {#1}
174   \__tag_struct_kid_mc_gput_right:nn
175   { \g__tag_struct_stack_current_tl }
176   {#1}
177   \prop_gput:Nxx \g__tag_mc_parenttree_prop
178   {#1}
179   { \g__tag_struct_stack_current_tl }
180 }

```

(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


```

181 \cs_new_protected:Npn \__tag_mc_bmc_artifact:
182 {
183   \__tag_mc_bmc:n {Artifact}
184 }
185 \cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
186 {
187   \__tag_mc_bdc:nn {Artifact}{/Type/#1}
188 }
189 \cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
190   % #1 is a var containing the artifact type
191 {
192   \tl_if_empty:NTF #1
193     { \__tag_mc_bmc_artifact: }
194     { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
195 }

```

(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.

```

196 \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:

```

197 \cs_new_protected:Npn \tag_mc_begin:n #1 %#1 keyval
198 {
199   \group_begin: %hm
200   \__tag_check_mc_if_nested:
201   \bool_gset_true:N \g__tag_in_mc_bool
202   \keys_set:nn { __tag / mc } {#1}
203   \bool_if:NTF \l__tag_mc_artifact_bool
204     { %handle artifact
205       \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
206     }
207     { %handle mcid type
208       \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
209       \__tag_mc_handle_mcid:VV
210         \l__tag_mc_key_tag_tl
211         \l__tag_mc_key_properties_tl
212       \tl_if_empty:NF {\l__tag_mc_key_label_tl}
213       {
214         \exp_args:NV
215         \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
216       }
217       \bool_if:NF \l__tag_mc_key_stash_bool
218       {
219         \__tag_mc_handle_stash:n { \int_use:N \c@g__tag_MCID_abs_int }
220       }
221     }
222   \group_end:

```

```

223 }
224 \cs_new_protected:Nn \tag_mc_end:
225 {
226   \__tag_check_mc_if_open:
227   \bool_gset_false:N \g__tag_in_mc_bool
228   \tl_gset:Nn \g__tag_mc_key_tag_tl { }
229   \__tag_mc_emc:
230 }

```

(End definition for `\tag_mc_begin:n` and `\tag_mc_end:`. These functions are documented on page 1.)

`\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???

```

231 \cs_new_protected:Npn \tag_mc_use:n #1 %#1: label name
232 {
233   \tl_set:Nx \l__tag_tmpa_tl { \__tag_ref_value:enn{tagpdf-#1}{tagmcabs}{ } }
234   \tl_if_empty:NTF\l__tag_tmpa_tl
235   {
236     \msg_warning:nnn {tag} {mc-label-unknown} {#1}
237   }
238   {
239     \prop_gput:Nxx
240     \g__tag_mc_parenttree_prop
241     {
242       \l__tag_tmpa_tl
243       %\__tag_ref_value:enn {tagpdf-#1} {tagmcabs} { }
244     }
245     {
246       \g__tag_struct_stack_current_tl
247     }
248     \__tag_struct_kid_mc_gput_right:nn
249     {
250       \g__tag_struct_stack_current_tl
251     }
252     {
253       \l__tag_tmpa_tl
254       %\__tag_ref_value:enn {tagpdf-#1} {tagmcabs} { }
255     }
256   }
257 }

```

(End definition for `\tag_mc_use:n`. This function is documented on page 1.)

4.3 Keys

Definitions are different in `luamode`. `tag` and `raw` are expanded as `\directlua` in `lua` does it too and we assume that their values are safe.

```

tag
raw
alttext
alttext-o
actualtext
actualtext-o
label
artifact
258 \keys_define:nn { __tag / mc }
259 {

```

```

260 tag .code:n = % the name (H,P,Span) etc
261 {
262     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
263     \tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
264 },
265 raw .code:n =
266 {
267     \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
268 },
269 alttext .code:n = % Alt property
270 {
271     \str_set_convert:Nnon
272     \l__tag_tmpa_str
273     { #1 }
274     { default }
275     { utf16/hex }
276     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
277     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
278 },
279 alttext-o .code:n = % Alt property
280 {
281     \str_set_convert:Noon
282     \l__tag_tmpa_str
283     { #1 }
284     { default }
285     { utf16/hex }
286     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
287     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
288 },
289 actualtext .code:n = % ActualText property
290 {
291     \str_set_convert:Nnon
292     \l__tag_tmpa_str
293     { #1 }
294     { default }
295     { utf16/hex }
296     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
297     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
298 },
299 actualtext-o .code:n = % ActualText property
300 {
301     \str_set_convert:Noon
302     \l__tag_tmpa_str
303     { #1 }
304     { default }
305     { utf16/hex }
306     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
307     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
308 },
309 label .tl_set:N = \l__tag_mc_key_label_tl,
310 artifact .code:n =
311 {
312     \exp_args:Nnx
313     \keys_set:nn

```

```

314         { __tag / mc }
315         { __artifact-bool, __artifact-type=#1 }
316     },
317     artifact .default:n    = {notype}
318 }
319 </generic>

```

(End definition for *tag* and others. These functions are documented on page 2.)

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 currently local, but this will probably change, see the `global-mc` option.

`\newattribute \l_@@_mc_type_attr`: the value represent the type

`\newattribute \l_@@_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 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.

```

320 (*luamode)
321 \ProvidesExplPackage {tagpdf-mc-code-lua} {2021-06-14} {0.82}
322 {tagpdf - mc code only for the luamode }

```

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

```

323 \hook_gput_code:nnn{begindocument}{tagpdf/mc}
324 {
325     \bool_if:NT\g__tag_active_mc_bool
326     {
327         \directlua
328         {
329             if~luatexbase.callbacktypes.pre_shipout_filter~then~
330                 luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
331                     ltx.__tag.func.mark_shipout(TAGBOX)~return~true~
332                 end, "tagpdf")~
333             end
334         }
335     }
336 }

```

```

335     \directlua
336     {
337         if~luatexbase.callbacktypes.pre_shipout_filter~then~
338             token.get_next()~
339             end
340     }~\@secondoftwo~\@gobble
341     {
342         \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
343         {
344             \directlua
345             { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
346         }
347     }
348 }
349 }

```

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: 350 \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
351 {
352     \int_compare:nNnTF { -2147483647 }={ \l__tag_mc_type_attr }
353     { \prg_return_false: }
354     { \prg_return_true: }
355 }
356
357 \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: 358 \cs_new:Nn \__tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
359 {
360     \__tag_attribute_set:Nn \l__tag_mc_type_attr
361     {
362         \directlua { ltx.__tag.func.output_num_from ("#1") }
363     }
364     \__tag_attribute_set:Nn \l__tag_mc_cnt_attr { \__tag_get_mc_abs_cnt: }
365 }
366
367 \cs_generate_variant:Nn \__tag_mc_lua_set_mc_type_attr:n { o }
368
369 \cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
370 {
371     \__tag_attribute_unset:N \l__tag_mc_type_attr
372     \__tag_attribute_unset:N \l__tag_mc_cnt_attr
373 }
374

```

(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 kids we need a variant for the case that it is the only kid, to get the array right

`__tag_mc_insert_mcid_single_kids:n`

```

375 \cs_new:Nn \__tag_mc_insert_mcid_kids:n
376 {
377   \directlua { ltx.__tag.func.mc_insert_kids (#1,0) }
378 }
379
380 \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
381 {
382   \directlua {ltx.__tag.func.mc_insert_kids (#1,1) }
383 }

```

(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

```

384 \cs_new:Nn \__tag_mc_handle_stash:n %1 mcidnum
385 {
386   \__tag_check_mc_used:n { #1 }
387   \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
388                     % so use the kernel command
389   { g__tag_struct_kids\g__tag_struct_stack_current_tl _seq }
390   {
391     \__tag_mc_insert_mcid_kids:n {#1}%
392   }
393   \directlua
394   {
395     ltx.__tag.func.store_struct_mcab
396     (
397       \g__tag_struct_stack_current_tl,#1
398     )
399   }
400   \prop_gput:Nxx
401   \g__tag_mc_parenttree_prop
402   { #1 }
403   { \g__tag_struct_stack_current_tl }
404 }
405
406 \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. Unlike the generic version there is currently no group as the attribute is set locally. This means one must be careful with the keys. If the attribute is global again, we can change this.

```

407 \cs_new_protected:Nn \tag_mc_begin:n
408 {
409   %\group_begin:
410   %\__tag_check_mc_if_nested:
411   %\bool_gset_true:N \g__tag_in_mc_bool
412   \bool_set_false:N\l__tag_mc_artifact_bool
413   \tl_clear:N \l__tag_mc_key_properties_tl
414   \int_gincr:N \c@g__tag_MCID_abs_int
415   \keys_set:nn { __tag / mc }{ label={}, #1 }
416   %check that a tag or artifact has been used
417   \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl

```

```

418 %set the attributes:
419 \__tag_mc_lua_set_mc_type_attr:o { \l__tag_mc_key_tag_tl }
420 \bool_if:NF \l__tag_mc_artifact_bool
421 { % store the absolute num name in a label:
422   \tl_if_empty:NF {\l__tag_mc_key_label_tl}
423   {
424     \exp_args:NV
425     \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
426   }
427   % if not stashed record the absolute number
428   \bool_if:NF \l__tag_mc_key_stash_bool
429   {
430     \exp_args:Nx \__tag_mc_handle_stash:n { \__tag_get_mc_abs_cnt: }
431   }
432 }
433 %\group_end:
434 }

```

(End definition for \tag_mc_begin:n. This function is documented on page 1.)

\tag_mc_end:
\tag_mc_use:n

```

435 \cs_new_protected:Nn \tag_mc_end:
436 {
437   %\__tag_check_mc_if_open:
438   %\bool_gset_false:N \g__tag_in_mc_bool
439   \bool_set_false:N\l__tag_mc_artifact_bool
440   \__tag_mc_lua_unset_mc_type_attr:
441   \tl_set:Nn \l__tag_mc_key_tag_tl { }
442 }
443
444 \cs_new_protected:Nn \tag_mc_use:n %\#1: label name
445 {
446   \tl_set:Nx \l__tag_tmpa_tl { \__tag_ref_value:enn{tagpdf-#1}{tagmcabs}{}} }
447   \tl_if_empty:NTF\l__tag_tmpa_tl
448   {
449     \msg_warning:nnn {tag} {mc-label-unknown} { #1 }
450   }
451   {
452     \__tag_mc_handle_stash:o { \l__tag_tmpa_tl }
453   }
454 }

```

(End definition for \tag_mc_end: and \tag_mc_use:n. These functions are documented on page 1.)

__tag_get_data_mc_tag: The command to retrieve the current mc tag. When we change to global this should use a global variable too!!

```

455 \cs_new:Npn \__tag_get_data_mc_tag: { \l__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	456 \keys_define:nn { __tag / mc }
alttext	
alttext-o	
actualtext	
actualtext-o	
label	
artifact	

```

457 {
458   tag .code:n = %
459     {%????????? \pdfescapename??
460     \tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
461     \directlua
462       {
463         ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag", "#1")
464       }
465   },
466   raw .code:n =
467     {
468       \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
469       \directlua
470         {
471           ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw", "#1")
472         }
473     },
474   alttext .code:n      = % Alt property
475   {
476     \str_set_convert:Nnon
477       \l__tag_tmpa_str
478       { #1 }
479       { default }
480       { utf16/hex }
481     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
482     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
483     \directlua
484       {
485         ltx.__tag.func.store_mc_data
486           (
487             \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
488           )
489       }
490   },
491   alttext-o .code:n      = % Alt property
492   {
493     \str_set_convert:Nnon
494       \l__tag_tmpa_str
495       { #1 }
496       { default }
497       { utf16/hex }
498     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
499     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
500     \directlua
501       {
502         ltx.__tag.func.store_mc_data
503           (
504             \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
505           )
506       }
507   },
508   actualtext .code:n      = % Alt property
509   {
510     \str_set_convert:Nnon

```



```

511         \l__tag_tmpa_str
512         { #1 }
513         { default }
514         { utf16/hex }
515     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
516     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
517     \directlua
518     {
519         ltx.__tag.func.store_mc_data
520         (
521             \__tag_get_mc_abs_cnt:,"actualtext","/ActualText~<\str_use:N \l__tag_tmpa_str
522         )
523     }
524 },
525 actualtext-o .code:n      = % Alt property
526 {
527     \str_set_convert:Noon
528     \l__tag_tmpa_str
529     { #1 }
530     { default }
531     { utf16/hex }
532     \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
533     \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
534     \directlua
535     {
536         ltx.__tag.func.store_mc_data
537         (
538             \__tag_get_mc_abs_cnt:,
539             "actualtext",
540             "/ActualText~<\str_use:N \l__tag_tmpa_str>"
541         )
542     }
543 },
544 label .code:n =
545 {
546     \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
547     \directlua
548     {
549         ltx.__tag.func.store_mc_data
550         (
551             \__tag_get_mc_abs_cnt:,"label","#1"
552         )
553     }
554 },
555 __artifact-store .code:n =
556 {
557     \directlua
558     {
559         ltx.__tag.func.store_mc_data
560         (
561             \__tag_get_mc_abs_cnt:,"artifact","#1"
562         )
563     }
564 },

```

```

565     artifact .code:n      =
566     {
567         \exp_args:Nnx
568         \keys_set:nn
569         { __tag / mc}
570         { __artifact-bool, __artifact-type=#1, tag=Artifact }
571         \exp_args:Nnx
572         \keys_set:nn
573         { __tag / mc }
574         { __artifact-store=\l__tag_mc_artifact_type_tl }
575     },
576     artifact .default:n    = { notype }
577 }
578
579 </luamode>

```

(End definition for `tag` and others. These functions are documented on page 2.)

Index

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

A		<code>\cs_new_protected:Nn</code> 224, 407, 435, 444
<code>actualtext</code> 2, 258, <u>456</u>		<code>\cs_new_protected:Npn</code> 22, 28, 33, 49, 117, 160, 165, 171, 181, 185, 189, 197, 231
<code>actualtext-o</code> 2, 258, <u>456</u>		<code>\cs_set_eq:NN</code> 113, 114, 115
<code>alttext</code> 2, 258, <u>456</u>		D
<code>alttext-o</code> 2, 258, <u>456</u>		<code>\directlua</code> 327, 335, 344, 362, 377, 382, 393, 461, 469, 483, 500, 517, 534, 547, 557
<code>artifact</code> 2, 258, <u>456</u>		E
artifact-bool internal commands:		exp commands:
<code>__artifact-bool</code> <u>66</u>		<code>\exp_args:Nnx</code> 312, 567, 571
artifact-type internal commands:		<code>\exp_args:Nv</code> 194, 214, 424
<code>__artifact-type</code> <u>66</u>		<code>\exp_args:Nx</code> 430
B		<code>\exp_not:n</code> 158
bool commands:		G
<code>\bool_gset_false:N</code> 227, 438		group commands:
<code>\bool_gset_true:N</code> 201, 411		<code>\group_begin:</code> 199, 409
<code>\bool_if:NTF</code> 106, 203, 217, 325, 420, 428		<code>\group_end:</code> 222, 433
<code>\bool_new:N</code> 12, 13, 100		H
<code>\bool_set_false:N</code> 412, 439		hook commands:
C		<code>\hook_gput_code:nmn</code> 323, 342
c@g internal commands:		I
<code>\c@g__tag_MCID_abs_int</code> 7, 119, 124, 153, 219, 414		int commands:
cs commands:		<code>\int_compare:nNnTF</code> 352
<code>\cs_generate_variant:Nn</code> 116, 170, 367, 406		
<code>\cs_new:Nn</code> 18, 196, 358, 369, 375, 380, 384		
<code>\cs_new:Npn</code> 7, 455		

\int_eval:n	141, 158	tag commands:	
\int_gincr:N	119, 414	\tag_get:n	37, 40
\int_gset:Nn	137	\tag_mc_artifact_group_begin:n	..
\int_gzero:N	145	1, 22, 22
\int_new:N	8	\tag_mc_artifact_group_end:	1, 22, 28
\int_use:N	7, 124, 153, 219	\tag_mc_begin:n
		1, 24, 59, 197, 197, 407, 407
K		\tag_mc_begin_pop:n	1, 33, 49
keys commands:		\tag_mc_end:	1, 31, 41, 197, 224, 435, 435
\keys_define:nn	66, 258, 456	\tag_mc_end_push:	1, 33, 33
\keys_set:nn	202, 313, 415, 568, 572	\tag_mc_if_in:	111, 350, 357
		\tag_mc_if_in:TF	2, 104
L		\tag_mc_if_in_p:	2, 104
label	2, 258, 456	\tag_mc_use:n	1, 231, 231, 435, 444
		\tag_stop_group_begin:	25
M		\tag_stop_group_end:	30
msg commands:		tag internal commands:	
\msg_warning:nnn	236, 449	\g__tag_active_mc_bool	325
		__tag_attribute_set:Nn	360, 364
N		__tag_attribute_unset:N	371, 372
\newcounter	6	__tag_check_mc_if_nested:	200, 410
		__tag_check_mc_if_open:	226, 437
P		__tag_check_mc_pushed_popped:nn
pdf commands:		38, 45, 55, 58, 63
\pdf_bdc:nn	115	__tag_check_mc_tag:N	208, 417
\pdf_bmc:n	113	__tag_check_mc_used:n	173, 386
\pdf_emc:	114	__tag_get_data_mc_tag:
\pdfescapename	459	196, 196, 455, 455
prg commands:		__tag_get_mc_abs_cnt:
\prg_new_conditional:Nnn	104, 350	7, 7, 364, 430,
\prg_new_eq_conditional:NNn	111, 357	463, 471, 487, 504, 521, 538, 551, 561
\prg_return_false:	108, 353	\g__tag_in_mc_bool
\prg_return_true:	107, 354	100, 106, 201, 227, 411, 438
prop commands:		\l__tag_mc_artifact_bool
\prop_get:NnNTF	129	12, 69, 203, 412, 420, 439
\prop_gput:Nnn	177, 239, 400	\l__tag_mc_artifact_type_tl
\ProvidesExplPackage	3, 98, 321	11, 73, 77, 81, 85, 89, 93, 205, 574
		__tag_mc_bdc:nn	112, 115, 116, 156, 187
R		__tag_mc_bdc_mcid:n	117, 160
raw	2, 258, 456	__tag_mc_bdc_mcid:nn
		117, 117, 162, 167
S		__tag_mc_bmc:n	112, 113, 183
seq commands:		__tag_mc_bmc_artifact:	181, 181, 193
\seq_gpop:NNTF	51	__tag_mc_bmc_artifact:n	181, 185, 194
\seq_gpush:Nn	37, 44	\l__tag_mc_cnt_attr	364, 372
\seq_gput_right:Nn	387	__tag_mc_emc:	112, 114, 229
\seq_new:N	10	__tag_mc_handle_artifact:N
stash	2, 66	181, 189, 205
str commands:		__tag_mc_handle_mc_label:n
\str_set_convert:Nnnn	18, 18, 215, 425
271, 281, 291, 301, 476, 493, 510, 527		__tag_mc_handle_mcid:nn
\str_use:N	487, 504, 521, 540	117, 165, 170, 209
		__tag_mc_handle_stash:n
T		171, 171, 219, 384, 384, 406, 430, 452
tag	2, 258, 456		

<code>__tag_mc_if_in:</code>	104, 111, 350, 350, 357
<code>__tag_mc_if_in:TF</code>	35, 104
<code>__tag_mc_if_in_p:</code>	104
<code>__tag_mc_insert_mcid_kids:n</code>	375, 375, 391
<code>__tag_mc_insert_mcid_single_-kids:n</code>	375, 380
<code>\l__tag_mc_key_label_tl</code>	14, 212, 215, 309, 422, 425, 546
<code>\l__tag_mc_key_properties_tl</code>	14, 211, 267, 276, 277, 286, 287, 296, 297, 306, 307, 413, 468, 481, 482, 498, 499, 515, 516, 532, 533
<code>\l__tag_mc_key_stash_bool</code>	12, 68, 217, 428
<code>\g__tag_mc_key_tag_tl</code>	14, 196, 228, 263
<code>\l__tag_mc_key_tag_tl</code>	14, 208, 210, 262, 417, 419, 441, 455, 460
<code>__tag_mc_lua_set_mc_type_attr:n</code>	358, 358, 367, 419
<code>__tag_mc_lua_unset_mc_type_attr:</code>	358, 369, 440
<code>\g__tag_mc_parenttree_prop</code>	9, 10, 177, 240, 401
<code>\l__tag_mc_ref_abspage_tl</code>	102, 120, 132, 140, 148
<code>\g__tag_mc_stack_seq</code>	10, 37, 44, 51
<code>\l__tag_mc_tmpa_tl</code>	103, 134, 137, 141
<code>\l__tag_mc_type_attr</code>	352, 360, 371
<code>g__tag_MCID_abs_int</code>	6
<code>\g__tag_MCID_byabspage_prop</code>	101, 130, 139, 147
<code>\g__tag_MCID_tmp_bypage_int</code>	8, 137, 145, 158
<code>__tag_prop_gput:Nnn</code>	138, 146
<code>__tag_prop_new:N</code>	9, 101
<code>__tag_ref_label:nn</code>	20, 151
<code>__tag_ref_value:nnn</code>	122, 233, 243, 254, 446
<code>__tag_struct_kid_mc_gput_-right:nn</code>	174, 248
<code>\g__tag_struct_stack_current_tl</code>	175, 179, 246, 250, 389, 397, 403
<code>\l__tag_tmpa_str</code>	272, 277, 282, 287, 292, 297, 302, 307, 477, 482, 487, 494, 499, 504, 511, 516, 521, 528, 533, 540
<code>\l__tag_tmpa_tl</code>	51, 53, 58, 59, 233, 234, 242, 253, 446, 447, 452
TeX and L ^A T _E X 2 _ε commands:	
<code>\@gobble</code>	340
<code>\@secondoftwo</code>	340
tl commands:	
<code>\tl_clear:N</code>	413
<code>\tl_gset:Nn</code>	228, 263
<code>\tl_if_empty:N</code>	192, 212, 234, 422, 447
<code>\tl_if_eq:NnTF</code>	53
<code>\tl_new:N</code>	11, 14, 15, 16, 17, 102, 103
<code>\tl_put_right:Nn</code>	267, 276, 277, 286, 287, 296, 297, 306, 307, 468, 481, 482, 498, 499, 515, 516, 532, 533
<code>\tl_set:Nn</code>	73, 77, 81, 85, 89, 93, 120, 233, 262, 441, 446, 460, 546