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

`\tag_mc_begin:n` `\tag_mc_begin:n{⟨key-values⟩}`
`\tag_mc_end:` `\tag_mc_end:`

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 if they are correctly nested and issue a warning if not.

`\tag_mc_use:n` `\tag_mc_use:n{⟨label⟩}`

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.

`\tag_mc_artifact_group_begin:n` `\tag_mc_artifact_group_begin:n {⟨name⟩}`
`\tag_mc_artifact_group_end:` `\tag_mc_artifact_group_end:`

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

`\tag_mc_end_push:` `\tag_mc_end_push:`
`\tag_mc_begin_pop:n` `\tag_mc_begin_pop:n{⟨key-values⟩}`

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 looses info like the alttext for now.

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

`\tag_mc_if_in_p:` ⋆ `\tag_mc_if_in:TF {⟨true code⟩} {⟨false code⟩}`
`\tag_mc_if_in:TF` ⋆ Determines if a mc-chunk is open.

2 Public keys

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

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

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

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

`alttext` `alttext-o` This key inserts an `/Alt` value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. With `alttext-o` the value is expanded once.

`actualtext` `actualtext-o` This key inserts an `/ActualText` value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded.With `actualtext-o` the value is expanded once.

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

`stash` 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 luamode and code not used by luamode.

3 Marked content code – shared

```
⟨@@=tag⟩
⟨*shared⟩
\ProvidesExplPackage {tagpdf-mc-code-shared} {2021-06-14} {0.82}
  {part of tagpdf - code related to marking chunks -
   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`

```
\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.

```
\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.

```
\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

```
\__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:

```
\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.

```
\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`

```
\bool_new:N \l__tag_mc_key_stash_bool
\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
\g__tag_mc_key_tag_tl
\l__tag_mc_key_label_tl
\l__tag_mc_key_properties_tl

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

```
\tl_new:N \l__tag_mc_key_tag_tl
\tl_new:N \g__tag_mc_key_tag_tl
\tl_new:N \l__tag_mc_key_label_tl
\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.

```
\cs_new:Nn \__tag_mc_handle_mc_label:n
  {
    \__tag_ref_label:en{tagpdf-#1}{mc}
  }
```

(*End definition for* `\__tag_mc_handle_mc_label:n`.)

\tag_mc_artifact_group_begin:n
\tag_mc_artifact_group_end:

This opens an artifact of the type given in the argument, and then stops all tagging. It creates a group.

```
\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1
 {
  \tag_mc_begin:n {artifact=#1}
  \tag_stop_group_begin:
 }

\cs_new_protected:Npn \tag_mc_artifact_group_end:
 {
  \tag_stop_group_end:
  \tag_mc_end:
 }
```

(*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

```
\cs_new_protected:Npn \tag_mc_end_push:
  {
    \__tag_mc_if_in:TF
      {
        \seq_gpush:Nx \g__tag_mc_stack_seq { \tag_get:n {mc_tag} }
        \__tag_check_mc_pushed_popped:nn
          { pushed }
          { \tag_get:n {mc_tag} }
        \tag_mc_end:
```

```
      }
      {
        \seq_gpush:Nn \g__tag_mc_stack_seq {-1}
        \__tag_check_mc_pushed_popped:nn { pushed }{-1}
      }
  }

\cs_new_protected:Npn \tag_mc_begin_pop:n #1
  {
    \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl
      {
        \tl_if_eq:NnTF \l__tag_tmpa_tl {-1}
          {
            \__tag_check_mc_pushed_popped:nn {popped}{-1}
          }
          {
            \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl}
            \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1}
          }
      }
      {
        \__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing}
      }
  }
```

(*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
__artifact-bool
__artifact-type

the two internal artifact keys are use to define the public `artifact`.

```
\keys_define:nn { __tag / mc }
  {
    stash                    .bool_set:N    = \l__tag_mc_key_stash_bool,
    __artifact-bool          .bool_set:N    = \l__tag_mc_artifact_bool,
    __artifact-type          .choice:,
    __artifact-type / pagination .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination }
      },
    __artifact-type / layout     .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }
      },
    __artifact-type / page       .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }
      },
    __artifact-type / background .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }
      },
```

```
    __artifact-type / notype     .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl {}
      },
    __artifact-type /      .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl {}
      },
  }
```

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

⟨/shared⟩

4 Marked content code – generic mode

⟨*generic⟩

```
\ProvidesExplPackage {tagpdf-mc-code-generic} {2021-06-14} {0.82}
 {part of tagpdf - code related to marking chunks - generic mode}
```

4.1 Variables

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

```
\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*⟩

```
\__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.

```
\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

```
\tl_new:N \l__tag_mc_tmpa_tl
```

(*End definition for* `\l__tag_mc_tmpa_tl`.)

4.2 Functions

__tag_mc_if_in_p:
__tag_mc_if_in:TF
\tag_mc_if_in_p:
\tag_mc_if_in:TF

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.

```
\prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
  {
    \bool_if:NTF \g__tag_in_mc_bool
      { \prg_return_true:  }
      { \prg_return_false: }
  }

\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
__tag_mc_emc:
__tag_mc_bdc:nn
__tag_mc_bdc:nx

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. 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.

```
% #1 tag, #2 properties
\cs_set_eq:NN \__tag_mc_bmc:n  \pdf_bmc:n
\cs_set_eq:NN \__tag_mc_emc:   \pdf_emc:
\cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
\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
__tag_mc_bdc_mcid:n
__tag_mc_handle_mcid:nn
__tag_mc_handle_mcid:VV

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.

```
\cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
  {
    \int_gincr:N \c@g__tag_MCID_abs_int
    \tl_set:Nx \l__tag_mc_ref_abspage_tl
      {
        \__tag_ref_value:enn %3 args
          {
            mcid-\int_use:N \c@g__tag_MCID_abs_int
          }
          { tagabspage }
          {-1}
      }
    \prop_get:NoNTF
      \g__tag_MCID_byabspage_prop
      {
        \l__tag_mc_ref_abspage_tl
      }
      \l__tag_mc_tmpa_tl
      {
        %key already present, use value for MCID and add 1 for the next
        \int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
```

```
        \__tag_prop_gput:Nxx
          \g__tag_MCID_byabspage_prop
          { \l__tag_mc_ref_abspage_tl }
          { \int_eval:n {\l__tag_mc_tmpa_tl +1} }
      }
      {
        %key not present, set MCID to 0 and insert 1
        \int_gzero:N \g__tag_MCID_tmp_bypage_int
        \__tag_prop_gput:Nxx
          \g__tag_MCID_byabspage_prop
          { \l__tag_mc_ref_abspage_tl }
          {1}
      }
    \__tag_ref_label:en
      {
        mcid-\int_use:N \c@g__tag_MCID_abs_int
      }
      { mc }
     \__tag_mc_bdc:nx
       {#1}
       { /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
 }
\cs_new_protected:Npn \__tag_mc_bdc_mcid:n #1
  {
    \__tag_mc_bdc_mcid:nn {#1} {}
  }

\cs_new_protected:Npn \__tag_mc_handle_mcid:nn #1 #2 %#1 tag, #2 properties
  {
    \__tag_mc_bdc_mcid:nn {#1} {#2}
  }

\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, ...

```
\cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
  {
    \__tag_check_mc_used:n {#1}
    \__tag_struct_kid_mc_gput_right:nn
      { \g__tag_struct_stack_current_tl }
      {#1}
   \prop_gput:Nxx \g__tag_mc_parenttree_prop
     {#1}
     { \g__tag_struct_stack_current_tl }
  }
```

(*End definition for* `\__tag_mc_handle_stash:n`.)

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

```
\cs_new_protected:Npn  \__tag_mc_bmc_artifact:
  {
    \__tag_mc_bmc:n {Artifact}
  }
\cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
  {
    \__tag_mc_bdc:nn {Artifact}{/Type/#1}
  }
\cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
   % #1 is a var containing the artifact type
  {
    \tl_if_empty:NTF #1
      { \__tag_mc_bmc_artifact: }
      { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
  }
```

(*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.

```
\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` `\tag_mc_end:` 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.

```
\cs_new_protected:Npn \tag_mc_begin:n #1 %#1 keyval
  {
    \group_begin: %hm
    \__tag_check_mc_if_nested:
    \bool_gset_true:N \g__tag_in_mc_bool
    \keys_set:nn { __tag / mc } {#1}
    \bool_if:NTF \l__tag_mc_artifact_bool
      { %handle artifact
        \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
      }
      { %handle mcid type
        \__tag_check_mc_tag:N  \l__tag_mc_key_tag_tl
        \__tag_mc_handle_mcid:VV
           \l__tag_mc_key_tag_tl
           \l__tag_mc_key_properties_tl
        \tl_if_empty:NF {\l__tag_mc_key_label_tl}
          {
            \exp_args:NV
            \__tag_mc_handle_mc_label:n \l__tag_mc_key_label_tl
          }
        \bool_if:NF \l__tag_mc_key_stash_bool
          {
            \__tag_mc_handle_stash:n { \int_use:N \c@g__tag_MCID_abs_int }
          }
      }
    \group_end:
```

```
  }
\cs_new_protected:Nn \tag_mc_end:
  {
    \__tag_check_mc_if_open:
    \bool_gset_false:N \g__tag_in_mc_bool
    \tl_gset:Nn  \g__tag_mc_key_tag_tl { }
    \__tag_mc_emc:
  }
```

(*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???

```
\cs_new_protected:Npn \tag_mc_use:n #1 %#1: label name
  {
    \tl_set:Nx  \l__tag_tmpa_tl { \__tag_ref_value:enn{tagpdf-#1}{tagmcabs}{} }
    \tl_if_empty:NTF\l__tag_tmpa_tl
      {
        \msg_warning:nnn {tag} {mc-label-unknown} {#1}
      }
      {
        \prop_gput:Nxx
          \g__tag_mc_parenttree_prop
          {
            \l__tag_tmpa_tl
            %\__tag_ref_value:enn {tagpdf-#1} {tagmcabs} {}
          }
          {
            \g__tag_struct_stack_current_tl
          }
        \__tag_struct_kid_mc_gput_right:nn
          {
            \g__tag_struct_stack_current_tl
          }
          {
            \l__tag_tmpa_tl
            %\__tag_ref_value:enn {tagpdf-#1} {tagmcabs} {}
          }
       }
  }
```

(*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`

```
\keys_define:nn { __tag / mc }
  {
```

```
    tag .code:n = % the name (H,P,Span) etc
      {
        \tl_set:Nx   \l__tag_mc_key_tag_tl { #1 }
        \tl_gset:Nx  \g__tag_mc_key_tag_tl { #1 }
      },
    raw  .code:n =
      {
        \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
      },
    alttext .code:n = % Alt property
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
        \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      },
    alttext-o .code:n      = % Alt property
      {
        \str_set_convert:Noon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
        \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      },
    actualtext .code:n      = % ActualText property
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
        \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      },
    actualtext-o .code:n      = % ActualText property
      {
        \str_set_convert:Noon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \tl_put_right:Nn \l__tag_mc_key_properties_tl { /ActualText~< }
        \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      },
    label .tl_set:N        = \l__tag_mc_key_label_tl,
    artifact .code:n       =
      {
        \exp_args:Nnx
          \keys_set:nn
```

```
            { __tag / mc }
            { __artifact-bool, __artifact-type=#1 }
      },
    artifact .default:n    = {notype}
  }
⟨/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.

```
⟨*luamode⟩
\ProvidesExplPackage {tagpdf-mc-code-lua} {2021-06-14} {0.82}
  {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.

```
\hook_gput_code:nnn{begindocument}{tagpdf/mc}
  {
    \bool_if:NT\g__tag_active_mc_bool
      {
        \directlua
          {
            if~luatexbase.callbacktypes.pre_shipout_filter~then~
              luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
              ltx.__tag.func.mark_shipout(TAGBOX)~return~true~
              end, "tagpdf")~
            end
          }
```

```
      \directlua
        {
          if~luatexbase.callbacktypes.pre_shipout_filter~then~
          token.get_next()~
          end
        }\@secondoftwo\@gobble
          {
            \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
              {
                \directlua
                  { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
              }
          }
    }
}
```

5.1 Commands

__tag_mc_if_in:
\tag_mc_if_in:

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

```
\prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
  {
    \int_compare:nNnTF { -2147483647 }={ \l__tag_mc_type_attr }
      { \prg_return_false:  }
      { \prg_return_true: }
  }

\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
__tag_mc_lua_set_mc_type_attr:o
__tag_mc_lua_unset_mc_type_attr:

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.

```
\cs_new:Nn \__tag_mc_lua_set_mc_type_attr:n % #1 is a tag name
  {
    \__tag_attribute_set:Nn \l__tag_mc_type_attr
      {
        \directlua { ltx.__tag.func.output_num_from ("#1") }
      }
    \__tag_attribute_set:Nn \l__tag_mc_cnt_attr  { \__tag_get_mc_abs_cnt: }
  }

\cs_generate_variant:Nn\__tag_mc_lua_set_mc_type_attr:n { o }

\cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
  {
    \__tag_attribute_unset:N \l__tag_mc_type_attr
    \__tag_attribute_unset:N \l__tag_mc_cnt_attr
  }

```

(*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
__tag_mc_insert_mcid_single_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

```
\cs_new:Nn \__tag_mc_insert_mcid_kids:n
  {
    \directlua { ltx.__tag.func.mc_insert_kids (#1,0) }
  }

\cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
  {
    \directlua {ltx.__tag.func.mc_insert_kids (#1,1) }
  }
```

(*End definition for* `\__tag_mc_insert_mcid_kids:n` *and* `\__tag_mc_insert_mcid_single_kids:n`.)

`\__tag_mc_handle_stash:n`
`\__tag_mc_handle_stash:o`

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

```
\cs_new:Nn \__tag_mc_handle_stash:n %1 mcidnum
  {
    \__tag_check_mc_used:n { #1 }
    \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
                       % so use the kernel command
      { g__tag_struct_kids_\g__tag_struct_stack_current_tl _seq }
      {
        \__tag_mc_insert_mcid_kids:n {#1}%
      }
    \directlua
      {
        ltx.__tag.func.store_struct_mcabs
          (
            \g__tag_struct_stack_current_tl,#1
          )
      }
    \prop_gput:Nxx
      \g__tag_mc_parenttree_prop
      { #1 }
      { \g__tag_struct_stack_current_tl }
  }

\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, wen can change this.

```
\cs_new_protected:Nn \tag_mc_begin:n
  {
    %\group_begin:
    %\__tag_check_mc_if_nested:
    %\bool_gset_true:N \g__tag_in_mc_bool
    \bool_set_false:N\l__tag_mc_artifact_bool
    \tl_clear:N \l__tag_mc_key_properties_tl
    \int_gincr:N \c@g__tag_MCID_abs_int
    \keys_set:nn { __tag / mc }{ label={}, #1 }
    %check that a tag or artifact has been used
    \__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
```

```
    %set the attributes:
    \__tag_mc_lua_set_mc_type_attr:o  { \l__tag_mc_key_tag_tl }
    \bool_if:NF \l__tag_mc_artifact_bool
      { % store the absolute num name in a label:
        \tl_if_empty:NF {\l__tag_mc_key_label_tl}
          {
            \exp_args:NV
             \__tag_mc_handle_mc_label:n  \l__tag_mc_key_label_tl
          }
       % if not stashed record the absolute number
        \bool_if:NF \l__tag_mc_key_stash_bool
          {
            \exp_args:Nx \__tag_mc_handle_stash:n { \__tag_get_mc_abs_cnt: }
          }
      }
    %\group_end:
  }
```

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

`\tag_mc_end:`
`\tag_mc_use:n`

```
\cs_new_protected:Nn \tag_mc_end:
  {
    %\__tag_check_mc_if_open:
    %\bool_gset_false:N \g__tag_in_mc_bool
    \bool_set_false:N\l__tag_mc_artifact_bool
    \__tag_mc_lua_unset_mc_type_attr:
    \tl_set:Nn  \l__tag_mc_key_tag_tl { }
  }

\cs_new_protected:Nn \tag_mc_use:n %#1: label name
  {
    \tl_set:Nx  \l__tag_tmpa_tl { \__tag_ref_value:enn{tagpdf-#1}{tagmcabs}{} }
    \tl_if_empty:NTF\l__tag_tmpa_tl
      {
        \msg_warning:nnn {tag} {mc-label-unknown} { #1 }
      }
      {
        \__tag_mc_handle_stash:o { \l__tag_tmpa_tl }
      }
  }
```

(*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!!

```
\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`
`raw`
`alttext`
`alttext-o`
`actualtext`
`actualtext-o`
`label`
`artifact`

TODO: check conversion, check if local/global setting is right.

```
\keys_define:nn { __tag / mc }
```

```
{
  tag .code:n = %
    {%%????????? \pdfescapename??
      \tl_set:Nx  \l__tag_mc_key_tag_tl { #1 }
      \directlua
        {
          ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","#1")
        }
    },
  raw .code:n =
    {
      \tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
      \directlua
        {
          ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
        }
    },
  alttext .code:n      = % Alt property
    {
      \str_set_convert:Nnon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
      \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      \directlua
        {
          ltx.__tag.func.store_mc_data
            (
               \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
            )
        }
    },
  alttext-o .code:n      = % Alt property
    {
      \str_set_convert:Noon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
      \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      \directlua
        {
          ltx.__tag.func.store_mc_data
            (
              \__tag_get_mc_abs_cnt:,"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
            )
        }
    },
  actualtext .code:n      = % Alt property
    {
      \str_set_convert:Nnon
```

```
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
      \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      \directlua
        {
          ltx.__tag.func.store_mc_data
            (
              \__tag_get_mc_abs_cnt:,"actualtext","/ActualText~<\str_use:N \l__tag_tmpa_str
            )
       }
    },
  actualtext-o .code:n      = % Alt property
    {
      \str_set_convert:Noon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
      \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }
      \directlua
        {
          ltx.__tag.func.store_mc_data
            (
              \__tag_get_mc_abs_cnt:,
              "actualtext",
              "/ActualText~<\str_use:N \l__tag_tmpa_str>"
            )
        }
    },
  label .code:n =
    {
      \tl_set:Nn\l__tag_mc_key_label_tl { #1 }
      \directlua
        {
          ltx.__tag.func.store_mc_data
            (
              \__tag_get_mc_abs_cnt:,"label","#1"
            )
        }
    },
  __artifact-store .code:n =
    {
      \directlua
        {
          ltx.__tag.func.store_mc_data
            (
              \__tag_get_mc_abs_cnt:,"artifact","#1"
            )
        }
    },
```

```
    artifact .code:n       =
      {
        \exp_args:Nnx
          \keys_set:nn
            { __tag / mc}
            { __artifact-bool, __artifact-type=#1, tag=Artifact }
        \exp_args:Nnx
          \keys_set:nn
            { __tag / mc }
            { __artifact-store=\l__tag_mc_artifact_type_tl }
      },
    artifact .default:n    = { notype }
  }

⟨/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.