tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer†

Released 2022-01-09

Contents

*This file describes v0.93, last revised 2022-01-09.

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

`\ref_value:nnn` `\ref_value:nnn{⟨label⟩}{⟨attribute⟩}{⟨fallback default⟩}`

This is a temporary definition which will have to move to l3ref. It allows to locally set a default value if the label or the attribute doesn't exist. See issue #4 in Accessible-xref.

`\tag_stop_group_begin:`
`\tag_stop_group_end:`

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

`activate-space␣(setup-key)`

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

`activate-mc␣(setup-key)`
`activate-tree␣(setup-key)`
`activate-struct␣(setup-key)`
`activate-all␣(setup-key)`

Keys to activate the various tagging steps

`no-struct-dest␣(setup-key)`

The key allows to suppress the creation of structure destinations

`log␣(setup-key)`

The `log` takes currently the values `none`, `v`, `vv`, `vvv`, `all`. More details are in `tagpdf-checks`.

`tagunmarked␣(setup-key)`

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

`tabsorder␣(setup-key)`

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

`tagstruct`
`tagstructobj`
`tagabspage`
`tagmcabs`
`tagmcid`

These are attributes used by the label/ref system.

1 Initialization and test if pdfmanagement is active.

```
⟨@@=tag⟩
⟨*package⟩
\ProvidesExplPackage {tagpdf} {2022-01-09} {0.93}
  { A package to experiment with pdf tagging }

\bool_if:nF
  {
    \bool_lazy_and_p:nn
      {\cs_if_exist_p:N \pdfmanagement_if_active_p:}
      { \pdfmanagement_if_active_p: }
  }
  {  %error for now, perhaps warning later.
    \PackageError{tagpdf}
     {
       PDF~resource~management~is~no~active!\MessageBreak
       tagpdf~will~no~work.
     }
     {
       Activate~it~with \MessageBreak
       \string\RequirePackage{pdfmanagement-testphase}\MessageBreak
       \string\DeclareDocumentMetadata{<options>}\MessageBreak
       before~\string\documentclass
     }
  }
⟨/package⟩
```

<*debug>

```
\ProvidesExplPackage {tagpdf-debug} {2022-01-09} {0.93}
  { debug code for tagpdf }
\@ifpackageloaded{tagpdf}{}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\endinp
    \end{macrocode}
⟨/debug⟩
% We map the internal module name \enquote{tag} to \enquote{tagpdf} in messages.
%    \begin{macrocode}
⟨*package⟩
\prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
⟨/package⟩
```

Debug mode has its special mapping:

```
⟨*debug⟩
\prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
\prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf~DEBUG}
⟨/debug⟩
```

2 Package options

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

```
⟨*package⟩
\bool_new:N\g__tag_mode_lua_bool
\DeclareOption {luamode}    { \sys_if_engine_luatex:T { \bool_gset_true:N \g__tag_mode_lua_bo
\DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
\ExecuteOptions{luamode}
```

```
\ProcessOptions
```

3 Packages

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

```
\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:

```
\cs_new_protected:Npn \__tag_lastpagelabel:
  {
    \legacy_if:nT { @filesw }
      {
        \exp_args:NNnx \exp_args:NNx\iow_now:Nn \@auxout
           {
             \token_to_str:N \newlabeldata
               {__tag_LastPage}
               {
                 {abspage} { \int_use:N \g_shipout_readonly_int}
                 {tagmcabs}{ \int_use:N \c@g__tag_MCID_abs_int }
               }
           }
      }
  }

\AddToHook{enddocument/afterlastpage}
 {\__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.

```
\cs_if_exist:NF \ref_value:nnn
  {
    \cs_new:Npn \ref_value:nnn #1#2#3
      {
        \exp_args:Nee
          \__ref_value:nnn
           { \tl_to_str:n {#1} } { \tl_to_str:n {#2} } {#3}
      }
    \cs_new:Npn \__ref_value:nnn #1#2#3
      {
        \tl_if_exist:cTF { g__ref_label_ #1 _ #2 _tl }
          { \tl_use:c { g__ref_label_ #1 _ #2 _tl } }
          {
            #3
```

```
            }
        }
    }
```

(*End definition for* `\ref_value:nnn`. *This function is documented on page 3.*)

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` `\l__tag_tmpa_box` `\l__tag_tmpb_box`

A few temporary variables

```
\tl_new:N    \l__tag_tmpa_tl
\str_new:N   \l__tag_tmpa_str
\prop_new:N  \l__tag_tmpa_prop
\seq_new:N   \l__tag_tmpa_seq
\seq_new:N   \l__tag_tmpb_seq
\clist_new:N \l__tag_tmpa_clist
\int_new:N   \l__tag_tmpa_int
\box_new:N   \l__tag_tmpa_box
\box_new:N   \l__tag_tmpb_box
```

(*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`

```
\clist_const:Nn \c__tag_refmc_clist     {tagabspage,tagmcabs,tagmcid}
\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.

```
\int_new:N  \l__tag_loglevel_int
```

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

`\g__tag_active_space_bool` `\g__tag_active_mc_bool` `\g__tag_active_tree_bool` `\g__tag_active_struct_bool` `\g__tag_active_struct_dest_bool`

These booleans should help to control the global behaviour of tagpdf. Ideally it should more or less do nothing if all are false. The space-boolean controles the interword space code, the mc-boolean activates `\tag_mc_begin:n`, the tree-boolean activates writing the finish code and the pdfmanagement related commands, the struct-boolean activates the storing of the structure data. In a normal document all should be active, the split is only there for debugging purpose. Structure destination will be activated automatically if pdf version 2.0 is detected, but with the boolean struct-dest-boolean one can suppress them. 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.

```
\bool_new:N \g__tag_active_space_bool
\bool_new:N \g__tag_active_mc_bool
\bool_new:N \g__tag_active_tree_bool
\bool_new:N \g__tag_active_struct_bool
\bool_new:N \g__tag_active_struct_dest_bool
\bool_gset_true:N \g__tag_active_struct_dest_bool
```

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

\l__tag_active_mc_bool
\l__tag_active_struct_bool

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

```
\bool_new:N \l__tag_active_mc_bool
\bool_set_true:N \l__tag_active_mc_bool
\bool_new:N \l__tag_active_struct_bool
\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.

```
\bool_new:N \g__tag_tagunmarked_bool
```

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

6 Variants of l3 commands

```
\prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F}
\cs_generate_variant:Nn \pdf_object_ref:n {e}
\cs_generate_variant:Nn \pdfannot_dict_put:nnn {nnx}
\cs_generate_variant:Nn \pdffile_embed_stream:nnn {nxx,oxx}
\cs_generate_variant:Nn \prop_gput:Nnn {Nxx}
\cs_generate_variant:Nn \prop_put:Nnn  {Nxx}
\cs_generate_variant:Nn \ref_label:nn { nv }
\cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
\cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
```

7 Setup label attributes

tagstruct
tagstructobj
tagabspage
tagmcabs
tagmcid

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`.

```
\ref_attribute_gset:nnnn { tagstruct } {0} { now }
  { \int_use:N \c@g__tag_struct_abs_int }
\ref_attribute_gset:nnnn { tagstructobj } {} { now }
  {
    \pdf_object_if_exist:eT {__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
      {
        \pdf_object_ref:e{__tag/struct/\int_use:N \c@g__tag_struct_abs_int}
      }
  }
\ref_attribute_gset:nnnn { tagabspage } {0} { shipout }
  { \int_use:N \g_shipout_readonly_int }
\ref_attribute_gset:nnnn { tagmcabs } {0} { now }
  { \int_use:N \c@g__tag_MCID_abs_int }
\ref_attribute_gset:nnnn {tagmcid }  {0} { now }
  { \int_use:N \g__tag_MCID_tmp_bypage_int }
```

(*End definition for* `tagstruct` *and others. These functions are documented on page 3.*)

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.

```
\cs_new_protected:Npn \__tag_ref_label:nn #1 #2 %#1 label, #2 name of list mc or struct
  {
    \@bsphack
    \ref_label:nv {#1}{c__tag_ref#2_clist}
    \@esphack
  }
\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 temporarly above.

```
\cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 %#1 label, #2 attribute, #3 default
  {
    \ref_value:nnn {#1}{#2}{#3}
  }
\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.

```
\cs_new:Npn \__tag_ref_value_lastpage:nn #1 #2
  {
    \ref_value:nnn {__tag_LastPage}{#1}{#2}
  }
```

(*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`
`\__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`

```
\cs_set_eq:NN \__tag_prop_new:N        \prop_new:N
\cs_set_eq:NN \__tag_seq_new:N         \seq_new:N
\cs_set_eq:NN \__tag_prop_gput:Nnn     \prop_gput:Nnn
\cs_set_eq:NN \__tag_seq_gput_right:Nn \seq_gput_right:Nn
\cs_set_eq:NN \__tag_seq_item:cn       \seq_item:cn
\cs_set_eq:NN \__tag_prop_item:cn      \prop_item:cn
\cs_set_eq:NN \__tag_seq_show:N        \seq_show:N
\cs_set_eq:NN \__tag_prop_show:N       \prop_show:N

\cs_generate_variant:Nn \__tag_prop_gput:Nnn      { Nxn , Nxx, Nnx , cnn, cxn, cnx, cno}
\cs_generate_variant:Nn \__tag_seq_gput_right:Nn  { Nx  , No, cn, cx }
```

```
\cs_generate_variant:Nn \__tag_prop_new:N   { c }
\cs_generate_variant:Nn \__tag_seq_new:N    { c }
\cs_generate_variant:Nn \__tag_seq_show:N   { c }
\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:`
`\tag_stop_group_end:`

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

```
\cs_new_protected:Npn \tag_stop_group_begin:
  {
    \group_begin:
    \bool_set_false:N \l__tag_active_struct_bool
    \bool_set_false:N \l__tag_active_mc_bool
  }
\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 3.*)

11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

`activate-space␣(setup-key)`
`activate-mc␣(setup-key)`
`activate-tree␣(setup-key)`
`activate-struct␣(setup-key)`
`activate-all␣(setup-key)`
`no-struct-dest␣(setup-key)`

Keys to (globally) activate tagging. `activate-space` activates the additional parsing needed for interword spaces. It is not documented, the parsing is currently implicitly activated by the known key `interwordspace`, as the code will perhaps move to some other place, now that it is better separated. `no-struct-dest` allows to suppress structure destinations.

```
\keys_define:nn { __tag / setup }
  {
    activate-space  .bool_gset:N = \g__tag_active_space_bool,
    activate-mc     .bool_gset:N = \g__tag_active_mc_bool,
    activate-tree   .bool_gset:N = \g__tag_active_tree_bool,
    activate-struct .bool_gset:N = \g__tag_active_struct_bool,
    activate-all    .meta:n =
      {activate-mc={#1},activate-tree={#1},activate-struct={#1}},
    activate-all  .default:n = true,
    no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,

```

(*End definition for* `activate-space (setup-key)` *and others. These functions are documented on page 3.*)

`log␣(setup-key)`

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

```
    log             .choice:,
    log / none      .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
    log / v         .code:n =
      {
```

```
      \int_set:Nn \l__tag_loglevel_int { 1 }
      \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:x {##1} }
    },
  log / vv        .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
  log / vvv       .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
  log / all       .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
```

(*End definition for* `log (setup-key)`. *This function is documented on page 3.*)

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

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

(*End definition for* `tagunmarked (setup-key)`. *This function is documented on page 3.*)

`tabsorder␣(setup-key)` This sets the tabsorder on a page. The values are `row`, `column`, `structure` (default) or `none`. Currently this is set more or less globally. More finer control can be added if needed.

```
  tabsorder       .choice:,
  tabsorder / row       .code:n =
    \pdfmanagement_add:nnn { Page } {Tabs}{/R},
  tabsorder / column    .code:n =
    \pdfmanagement_add:nnn { Page } {Tabs}{/C},
  tabsorder / structure .code:n =
    \pdfmanagement_add:nnn { Page } {Tabs}{/S},
  tabsorder / none      .code:n =
    \pdfmanagement_remove:nn {Page} {Tabs},
  tabsorder       .initial:n = structure,
  uncompress      .code:n = { \pdf_uncompress:  },
}
```

(*End definition for* `tabsorder (setup-key)`. *This function is documented on page 3.*)

12 loading of engine/more dependent code

```
\sys_if_engine_luatex:T
  {
    \file_input:n {tagpdf-luatex.def}
  }
⟨/package⟩
⟨*mcloading⟩
\bool_if:NTF \g__tag_mode_lua_bool
  {
   \RequirePackage {tagpdf-mc-code-lua}
  }
  {
   \RequirePackage {tagpdf-mc-code-generic} %
  }
⟨/mcloading⟩
⟨*debug⟩
\bool_if:NTF \g__tag_mode_lua_bool
  {
```

```
219     \RequirePackage {tagpdf-debug-lua}
220    }
221    {
222     \RequirePackage {tagpdf-debug-generic} %
223    }
224 ⟨/debug⟩
```

Part I
The tagpdf-checks module
Messages and check code
Part of the tagpdf package

1 Commands

`\tag_if_active_p:` ⋆
`\tag_if_active:TF` ⋆

This command tests if tagging is active. It only gives true if all tagging has been activated, *and* if tagging hasn't been stopped locally.

`\tag_get:n` ⋆ `\tag_get:n{⟨keyword⟩}`

This is a generic command to retrieve data. Currently the only sensible values for the argument ⟨*keyword*⟩ are `mc_tag` and `struct_tag`.

2 Description of log messages

2.1 `\ShowTagging` command

Argument	type	note
`\ShowTagging`mc-data = num	log+term	lua-only
`\ShowTagging`mc-current	log+term	
`\ShowTagging`struck-stack= [log|show]	log or term+stop	

2.2 Messages in checks and commands

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

2.3 Messages from the ptagging code

A few messages are issued in generic mode from the code which reinserts missing TMB/TME. This is currently done if log-level is larger than zero. TODO: reconsider log-level and messages when this code settles down.

2.4 Warning messages from the lua-code

The messages are triggered if the log-level is at least equal to the number.

message	log-level	remark
WARN TAG-NOT-TAGGED:	1	
WARN TAG-OPEN-MC:	1	
WARN SHIPOUT-MC-OPEN:	1	
WARN SHIPOUT-UPS:	0	shouldn't happen
WARN TEX-MC-INSERT-MISSING:	0	shouldn't happen
WARN TEX-MC-INSERT-NO-KIDS:	2	e.g. from empty hbox

2.5 Info messages from the lua-code

The messages are triggered if the log-level is at least equal to the number. `TAG` messages are from the traversing function, `TEX` from code used in the tagpdf-mc module. `PARENTREE` is the code building the parenttree.

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

message	log-level	remark
INFO TEX-MC-INSERT-KID-TEST	4	
INFO TEX-MC-INTO-STRUCT	3	
INFO TEX-STORE-MC-DATA	3	
INFO TEX-STORE-MC-KID	3	
INFO PARENTTREE-CHUNKS	3	
INFO PARENTTREE-NO-DATA	3	
INFO PARENTTREE-NUM	3	
INFO PARENTTREE-NUMENTRY	3	
INFO PARENTTREE-STRUCT-OBJREF	4	

2.6 Debug mode messages and code

If the package tagpdf-debug is loaded a number of commands are redefined and enhanced with additional commands which can be used to output debug messages or collect statistics. The commands are present but do nothing if the log-level is zero.

command	name	action	remark
\tag_mc_begin:n	mc-begin-insert	msg	
	mc-begin-ignore	msg	if inactive

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-checks-code} {2022-01-09} {0.93}
 {part of tagpdf - code related to checks, conditionals, debugging and messages}
⟨/header⟩
```

3 Messages

3.1 Messages related to mc-chunks

mc-nested This message is issue is 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.

```
⟨*package⟩
\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

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

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

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

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

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

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

```
\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.

```
\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.

`mc-popped`

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

```
\msg_new:nnn { tag } {mc-current}
  { current~MC:~
    \bool_if:NTF\g__tag_in_mc_bool
      {abscnt=\__tag_get_mc_abs_cnt:,~tag=\g__tag_mc_key_tag_tl}
      {no~MC~open,~current~abscnt=\__tag_get_mc_abs_cnt:"}
  }
```

(*End definition for* `mc-current`. *This function is documented on page* ??.)

3.2 Messages related to mc-chunks

`struct-no-objnum` Should not happen ...

```
\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.

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

```
\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`

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

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

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

```
\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`

`role-unknown-tag`

```
\msg_new:nnn { tag } {role-missing}     { tag~#1~has~no~role~assigned  }
\msg_new:nnn { tag } {role-unknown}     { role~#1~is~not~known  }
\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`

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

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

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

`sys-no-interwordspace` Currently only pdflatex and lualatex have some support for real spaces.

```
\msg_new:nnn { tag } {sys-no-interwordspace}
  {engine/output~mode~#1~doesn't~support~the~interword~spaces}
```

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

`\__tag_check_typeout_v:n` A simple logging function. By default is gobbles its argument, but the log-keys sets it to typeout.

```
\cs_set_eq:NN \__tag_check_typeout_v:n \use_none:n
```

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

para-hook-count-wrong

At the end of the document we check if the count of para-begin and para-end is identical. If not we issue a warning: this is normally a coding error and and breaks the structure.

```
\msg_new:nnnn { tag } {para-hook-count-wrong}
  {The~number~of~automatic~begin~(#1)~and~end~(#2)~para~hooks~differ!}
  {This~quite~probably~a~coding~error~and~the~structure~will~be~wrong!}
```

(*End definition for* `para-hook-count-wrong`. *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`.

```
\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:
\tag_if_active:*TF*

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

```
\prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
  {
     \bool_lazy_all:nTF
       {
         {\g__tag_active_struct_bool}
         {\g__tag_active_mc_bool}
         {\g__tag_active_tree_bool}
         {\l__tag_active_struct_bool}
         {\l__tag_active_mc_bool}
       }
       {
         \prg_return_true:
       }
       {
         \prg_return_false:
       }
  }
```

(*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
__tag_check_if_active_struct:TF

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.

```
\prg_new_conditional:Npnn \__tag_check_if_active_mc: {T,F,TF}
  {
    \bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
      {
         \prg_return_true:
      }
      {
         \prg_return_false:
      }
  }
\prg_new_conditional:Npnn \__tag_check_if_active_struct: {T,F,TF}
  {
    \bool_lazy_and:nnTF { \g__tag_active_struct_bool } { \l__tag_active_struct_bool }
      {
         \prg_return_true:
      }
      {
         \prg_return_false:
      }
  }
```

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

6.2 Checks related to stuctures

__tag_check_structure_has_tag:n

Structures must have a tag, so we check if the S entry is in the property. It is an error if this is missing. The argument is a number. The tests for existence and type is split in structures, as the tags are stored differently to the mc case.

```
\cs_new_protected:Npn \__tag_check_structure_has_tag:n #1 %#1 struct num
  {
    \prop_if_in:cnF { g__tag_struct_#1_prop }
      {S}
      {
        \msg_error:nn { tag } {struct-missing-tag}
      }
  }
```

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

__tag_check_structure_tag:N

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

```
\cs_new_protected:Npn \__tag_check_structure_tag:N #1
  {
    \prop_if_in:NoF \g__tag_role_tags_prop {#1}
      {
        \msg_warning:nnx { tag } {role-unknown-tag} {#1}
      }
  }
```

(*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.

```
\cs_new_protected:Npn \__tag_check_info_closing_struct:n #1 %#1 struct num
  {
    \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
      {
        \msg_info:nnn { tag } {struct-show-closing} {#1}
      }
  }

\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.

```
\cs_new_protected:Npn \__tag_check_no_open_struct:
  {
    \msg_error:nn { tag } {struct-faulty-nesting}
  }
```

(*End definition for* `\__tag_check_no_open_struct:`.)

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

```
\cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
  {
    \prop_get:cnNT
      {g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
      {P}
      \l_tmpa_tl
      {
        \msg_warning:nnn { tag } {struct-used-twice} {#1}
      }
  }
```

(*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.

```
\cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
  {
    \tl_if_empty:nTF {#2}
      {
        \msg_warning:nnn { tag } {role-missing} {#1}
      }
      {
        \prop_get:NnNTF \g__tag_role_tags_prop {#2} \l_tmpa_tl
          {
            \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
              {
                \msg_info:nnnn { tag } {role-tag} {#1} {#2}
              }
          }
          {
```

```
            \msg_warning:nnn { tag } {role-unknown} {#2}
          }
      }
  }
```

(*End definition for* `\__tag_check_add_tag_role:nn`.)

6.4 Check related to mc-chunks

`\__tag_check_mc_if_nested:`
`\__tag_check_mc_if_open:`

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

```
\cs_new_protected:Npn \__tag_check_mc_if_nested:
  {
    \__tag_mc_if_in:T
      {
        \msg_warning:nnx { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
      }
  }

\cs_new_protected:Npn \__tag_check_mc_if_open:
  {
    \__tag_mc_if_in:F
      {
        \msg_warning:nnx { tag } {mc-not-open} { \__tag_get_mc_abs_cnt: }
      }
  }
```

(*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.

```
\cs_new_protected:Npn \__tag_check_mc_pushed_popped:nn #1 #2
  {
    \int_compare:nNnT
      { \l__tag_loglevel_int } ={ 2 }
      { \msg_info:nnx {tag}{mc-#1}{#2} }
    \int_compare:nNnT
      { \l__tag_loglevel_int } > { 2 }
      {
        \msg_info:nnx {tag}{mc-#1}{#2}
        \seq_log:N \g__tag_mc_stack_seq
      }
  }
```

(*End definition for* `\__tag_check_mc_pushed_popped:nn`.)

`\__tag_check_mc_tag:N`

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

```
\cs_new_protected:Npn \__tag_check_mc_tag:N #1  %#1 is var with a tag name in it
  {
    \tl_if_empty:NT #1
      {
        \msg_error:nnx { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
      }
```

```
    \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
      {
        \msg_warning:nnx { tag } {role-unknown-tag} {#1}
      }
  }
```

(*End definition for* `\__tag_check_mc_tag:N`.)

`\g__tag_check_mc_used_intarray`
`\__tag_check_init_mc_used:`

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

```
\cs_new_protected:Npn \__tag_check_init_mc_used:
  {
    \intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
    \cs_gset_eq:NN \__tag_check_init_mc_used: \prg_do_nothing:
  }
```

(*End definition for* `\g__tag_check_mc_used_intarray` *and* `\__tag_check_init_mc_used:`.)

`\__tag_check_mc_used:n`

This checks if a mc is used twice.

```
\cs_new_protected:Npn \__tag_check_mc_used:n #1 %#1 mcid abscnt
  {
    \int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
      {
        \__tag_check_init_mc_used:
        \intarray_gset:Nnn \g__tag_check_mc_used_intarray
          {#1}
          { \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
        \int_compare:nNnT
          {
            \intarray_item:Nn \g__tag_check_mc_used_intarray {#1}
          }
          >
          { 1 }
          {
            \msg_warning:nnn { tag } {mc-used-twice} {#1}
          }
      }
  }
```

(*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.

```
\cs_new_protected:Npn \__tag_check_show_MCID_by_page:
  {
    \tl_set:Nx \l__tag_tmpa_tl
      {
        \__tag_ref_value_lastpage:nn
          {abspage}
          {-1}
```

```
    }
  \int_step_inline:nnnn {1}{1}
    {
      \l__tag_tmpa_tl
    }
    {
      \seq_clear:N \l_tmpa_seq
      \int_step_inline:nnnn
        {1}
        {1}
        {
          \__tag_ref_value_lastpage:nn
            {tagmcabs}
            {-1}
        }
        {
          \int_compare:nT
            {
              \__tag_ref_value:enn
                {mcid-####1}
                {tagabspage}
                {-1}
              =
              ##1
            }
            {
              \seq_gput_right:Nx \l_tmpa_seq
                {
                  Page##1-####1-
                  \__tag_ref_value:enn
                    {mcid-####1}
                    {tagmcid}
                    {-1}
                }
            }
        }
        \seq_show:N \l_tmpa_seq
    }
}
```

(*End definition for* `\__tag_check_show_MCID_by_page:`.)

6.5 Checks related to the state of MC on a page or in a split stream

The following checks are currently only usable in generic mode as they rely on the marks defined in the mc-generic module. They are used to detect if a mc-chunk has been split by a page break or similar and additional end/begin commands are needed.

`\__tag_check_mc_in_galley_p:`
`\__tag_check_mc_in_galley:`*TF*

At first we need a test to decide if `\tag_mc_begin:n` (tmb) and `\tag_mc_end:` (tme) has been used at all on the current galley. As each command issues two slightly different marks we can do it by comparing firstmarks and botmarks. The test assumes that

the marks have been already mapped into the sequence with `\@@_mc_get_marks:`. As `\seq_if_eq:NNTF` doesn't exist we use the tl-test.

```
\prg_new_conditional:Npnn \__tag_check_if_mc_in_galley: { T,F,TF }
 {
   \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq
    { \prg_return_false: }
    { \prg_return_true: }
 }
```

(*End definition for* `\__tag_check_mc_in_galley:TF`.)

`\__tag_check_if_mc_tmb_missing_p:`
`\__tag_check_if_mc_tmb_missing:TF`

This checks if a extra top mark ("extra-tmb") is needed. According to the analysis this the case if the firstmarks start with `e-` or `b+`. Like above we assume that the marks content is already in the seq's.

```
\prg_new_conditional:Npnn \__tag_check_if_mc_tmb_missing: { T,F,TF }
 {
  \bool_if:nTF
    {
      \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{e-}
      ||
      \str_if_eq_p:ee {\seq_item:Nn \l__tag_mc_firstmarks_seq {1}}{b+}
    }
    { \prg_return_true: }
    { \prg_return_false: }
 }
```

(*End definition for* `\__tag_check_if_mc_tmb_missing:TF`.)

`\__tag_check_if_mc_tme_missing_p:`
`\__tag_check_if_mc_tme_missing:TF`

This checks if a extra bottom mark ("extra-tme") is needed. According to the analysis this the case if the botmarks starts with `b+`. Like above we assume that the marks content is already in the seq's.

```
\prg_new_conditional:Npnn \__tag_check_if_mc_tme_missing: { T,F,TF }
 {
   \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+}
    { \prg_return_true: }
    { \prg_return_false: }
 }
```

(*End definition for* `\__tag_check_if_mc_tme_missing:TF`.)

```
⟨/package⟩
⟨*debug⟩
```

Code for tagpdf-debug. This will probably change over time. At first something for the mc commands.

```
\msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_li
\msg_new:nnn { tag / debug } {mc-end}   { MC~end~#1~[\msg_line_context:] }

\cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 }
     }
```

```
 }
\cs_new_protected:Npn \__tag_debug_mc_begin_ignore:n #1
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnnn { tag / debug } {mc-begin } {ignored} { #1 }
     }
 }
\cs_new_protected:Npn \__tag_debug_mc_end_insert:
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnn { tag / debug } {mc-end} {inserted}
     }
 }
\cs_new_protected:Npn \__tag_debug_mc_end_ignore:
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnn { tag / debug } {mc-end } {ignored}
     }
 }
```

And now something for the structures

```
\msg_new:nnn { tag / debug } {struct-begin}
  {
    Struct~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_context:]
  }
\msg_new:nnn { tag / debug } {struct-end}
  {
    Struct~end~#1~[\msg_line_context:]
  }

\cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
        \seq_log:N \g__tag_struct_tag_stack_seq
     }
 }
\cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
     }
 }
\cs_new_protected:Npn \__tag_debug_struct_end_insert:
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnn { tag / debug } {struct-end} {inserted}
        \seq_log:N \g__tag_struct_tag_stack_seq
     }
```

```
 }
\cs_new_protected:Npn \__tag_debug_struct_end_ignore:
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnn { tag / debug } {struct-end } {ignored}
     }
 }
```
⟨/debug⟩

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.

T

U