tagpdf – A package to experiment with pdf tagging[*]

Ulrike Fischer[†]

Released 2021-07-03

Contents

[*]This file describes v0.91, last revised 2021-07-03.
[†]E-mail: fischer@troubleshooting-tex.de

1 Initialization and test if pdfmanagement is active.

```
⟨@@=tag⟩
⟨*package⟩
\ProvidesExplPackage {tagpdf} {2021-07-03} {0.91}
  { 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
     }
  }
```

We map the internal module name "tag" to "tagpdf" in messages.

```
\prop_if_exist:NT \g_msg_module_name_prop
  {
    \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
  }
```

2 Package options

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

```
\bool_new:N\g__tag_mode_lua_bool
\DeclareOption {luamode}    { \sys_if_engine_luatex:T { \bool_gset_true:N \g__tag_mode_lua_boo
\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. See issue #4 in Accessible-xref.

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

```
\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* **??**.)

5 Variables

\l__tag_tmpa_tl
\l__tag_tmpa_str
\l__tag_tmpa_prop
\l__tag_tmpa_seq
\l__tag_tmpb_seq
\l__tag_tmpa_clist
\l__tag_tmpa_int

A few temporary variables

```
\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
```

(*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`

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. 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
```

(*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* **??**.)

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 temporarily 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* **??***.*)

11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

`activate-space` `activate-mc` `activate-tree` `activate-struct` `activate-all`

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.

```
\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,activate-tree,activate-struct},

```

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

`log`

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

```
    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 }},
    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`*. This function is documented on page* **??***.*)

`tagunmarked`

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

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

(*End definition for* `tagunmarked`*. This function is documented on page* **??***.*)

`tabsorder`

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

```
    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`*. This function is documented on page* **??**.)

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⟩
```

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

2.3 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

2.4 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. `PARENTTREE` 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	
`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	

message	log-level	remark
`INFO PARENTTREE-NUM`	3	
`INFO PARENTTREE-NUMENTRY`	3	
`INFO PARENTTREE-STRUCT-OBJREF`	4	

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-checks-code} {2021-07-03} {0.91}
 {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`
`mc-popped` Informational messages about mc-pushing.

```
\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 25.*)

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
role-unknown
role-unknown-tag

Warning message if either the tag or the role is missing

```
\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
new-tag

Info messages.

```
\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* ??*.*)

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* 13*.*)

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 13.*)

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:`.)

```
⟨/package⟩
```

Part II

The **tagpdf-user** module Code related to LaTeX2e user commands and document commands Part of the tagpdf package

1 Setup commands

`\tagpdfsetup` `\tagpdfsetup{⟨key val list⟩}`

This is the main setup command to adapt the behaviour of tagpdf. It can be used in the preamble and in the document (but not all keys make sense there).

2 Commands related to mc-chunks

```
\tagmcbegin \tagmcbegin {⟨key-val⟩}
\tagmcend   \tagmcend
\tagmcuse   \tagmcuse{⟨label⟩}
```

These are wrappers around `\tag_mc_begin:n`, `\tag_mc_end:` and `\tag_mc_use:n`. The commands and their argument are documentated in the tagpdf-mc module. In difference to the expl3 commands, `\tagmcbegin` issues also an `\ignorespaces`, and `\tagmcend` will issue in horizontal mode an `\unskip`.

`\tagmcifin` `\tagmcifin {⟨true code⟩}{⟨false code⟩}`

This is a wrapper around `\tag_mc_if_in:TF`. and tests if an mc is open or not. It is mostly of importance for pdflatex as lualatex doesn't mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

The command is probably not of much use and will perhaps disappear in future versions. It normally makes more sense to push/pop an mc-chunk.

3 Commands related to structures

```
\tagstructbegin \tagstructbegin {⟨key-val⟩}
\tagstructend   \tagstructend
\tagstructuse   \tagstructuse{⟨label⟩}
```

These are direct wrappers around `\tag_struct_begin:n`, `\tag_struct_end:` and `\tag_struct_use:n`. The commands and their argument are documentated in the tagpdf-struct module.

4 Debugging

`\ShowTagging` `\ShowTagging {⟨key-val⟩}`

This is a generic function to output various debugging helps. It not necessary stops the compilation. The keys and their function are described below.

`mc-data` `mc-data = ⟨number⟩`

This key is (currently?) relevant for lua mode only. It shows the data of all mc-chunks created so far. It is accurate only after shipout (and perhaps a second compilation), so typically should be issued after a newpage. The value is a positive integer and sets the first mc-shown. If no value is given, 1 is used and so all mc-chunks created so far are shown.

`mc-current` `mc-current`

This key shows the number and the tag of the currently open mc-chunk. If no chunk is open it shows only the state of the abs count. It works in all mode, but the output in luamode looks different.

`struct-stack` `struct-stack = log|show`

This key shows the current structure stack. With `log` the info is only written to the log-file, `show` stops the compilation and shows on the terminal. If no value is used, then the default is `show`.

5 Extension commands

The following commands and code parts are not core command of tagpdf. They either provide work-arounds for missing functionality elsewhere, or do a first step to apply tagpdf commands to document commands.

The commands and keys should be view as experimental!

This part will be regularly revisited to check if the code should go to a better place or can be improved and so can change easily.

5.1 Fake space

`\pdffakespace` (lua-only) This provides a lua-version of the `\pdffakespace` primitive of pdftex.

5.2 Paratagging

This is a first try to make use of the new paragraph hooks in a current LaTeX to automate the tagging of paragraph. It requires sane paragraph nesting, faulty code, e.g. a missing `\par` at the end of a low-level vbox can highly confuse the tagging. The tags should be carefully checked if this is used.

```
paratagging        paratagging = true|false
paratagging-show   paratagging-show = true|false
```

This keys can be used in `\tagpdfsetup` and enable/disable paratagging. `paratagging-show` puts small red numbers at the begin and end of a paragraph. This is meant as a debugging help. The number are boxes and have a (tiny) height, so they can affect typesetting.

```
\tagpdfparaOn
\tagpdfparaOff
```

These commands allow to enable/disable para tagging too and are a bit faster then `\tagpdfsetup`. But I'm not sure if the names are good.

5.3 Link tagging

Links need a special structure and cross reference system. This is added through hooks of the l3pdfannot module and will work automatically if tagging is activated.

Links should (probably) have an alternative text in the Contents key. It is unclear which text this should be and how to get it. Currently the code simply adds the fix texts `url` and `ref`. Another text can be added by changing the dictionary value:

```
\pdfannot_dict_put:nnn
{ link/GoTo }
{ Contents }
{ (ref) }
```

6 User commands and extensions of document commands

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-user} {2021-07-03} {0.91}
  {tagpdf - user commands}
⟨/header⟩
```

7 Setup and preamble commands

`\tagpdfsetup`

```
⟨*package⟩
\NewDocumentCommand \tagpdfsetup { m }
  {
    \keys_set:nn { __tag / setup } { #1 }
  }
```

(*End definition for* `\tagpdfsetup`. *This function is documented on page 24.*)

8 Commands for the mc-chunks

`\tagmcbegin`
`\tagmcend`
`\tagmcuse`

```
\NewDocumentCommand \tagmcbegin { m }
  {
    \tag_mc_begin:n {#1}%\ignorespaces
  }


\NewDocumentCommand \tagmcend {  }
  {
    %\if_mode_horizontal: \unskip \fi: %
    \tag_mc_end:
  }

\NewDocumentCommand \tagmcuse { m }
  {
    \tag_mc_use:n {#1}
  }

```

(*End definition for* `\tagmcbegin`*,* `\tagmcend`*, and* `\tagmcuse`*. These functions are documented on page 24.*)

`\tagmcifinTF` This is a wrapper around `\tag_mc_if_in:` and tests if an mc is open or not. It is mostly of importance for pdflatex as lualatex doesn't mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

```
\NewDocumentCommand \tagmcifinTF { m m }
  {
    \tag_mc_if_in:TF { #1 } { #2 }
  }
```

(*End definition for* `\tagmcifinTF`*. This function is documented on page* **??***.*)

9 Commands for the structure

`\tagstructbegin`
`\tagstructend`
`\tagstructuse`

These are structure related user commands. There are direct wrapper around the expl3 variants.

```
\NewDocumentCommand \tagstructbegin { m }
  {
    \tag_struct_begin:n {#1}
  }

\NewDocumentCommand \tagstructend {  }
  {
   \tag_struct_end:
  }

\NewDocumentCommand \tagstructuse { m }
  {
    \tag_struct_use:n {#1}
  }
```

(*End definition for* `\tagstructbegin`*,* `\tagstructend`*, and* `\tagstructuse`*. These functions are documented on page 24.*)

`\tagpdfifluatexTF`
`\tagpdfifluatexT`
`\tagpdfifpdftexTF`

I should deprecate them ...

```
\cs_set_eq:NN\tagpdfifluatexTF \sys_if_engine_luatex:TF
\cs_set_eq:NN\tagpdfifluatexT  \sys_if_engine_luatex:T
\cs_set_eq:NN\tagpdfifpdftexT  \sys_if_engine_pdftex:T
```

(*End definition for* `\tagpdfifluatexTF`*,* `\tagpdfifluatexT`*, and* `\tagpdfifpdftexTF`*. These functions are documented on page* **??**.)

10 Debugging

`\ShowTagging` This is a generic command for various show commands. It takes a keyval list, the various keys are implemented below.

```
\NewDocumentCommand\ShowTagging { m }
  {
    \keys_set:nn { __tag / show }{ #1}

  }
```

(*End definition for* `\ShowTagging`*. This function is documented on page 25.*)

`mc-data` This key is (currently?) relevant for lua mode only. It shows the data of all mc-chunks created so far. It is accurate only after shipout, so typically should be issued after a newpage. With the optional argument the minimal number can be set.

```
\keys_define:nn { __tag / show }
  {
    mc-data .code:n =
      {
        \sys_if_engine_luatex:T
          {
            \lua_now:e{ltx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
          }
      }
    ,mc-data .default:n = 1
  }

```

(*End definition for* `mc-data`*. This function is documented on page 25.*)

`mc-current` This shows some info about the current mc-chunk. It works in generic and lua-mode.

```
\keys_define:nn { __tag / show }
  { mc-current .code:n =
     {
       \bool_if:NTF \g__tag_mode_lua_bool
         {
           \sys_if_engine_luatex:T
             {
               \int_compare:nNnTF
                 { -2147483647 }
                  =
                 {
                   \lua_now:e
                     {
                        tex.print
                         (tex.getattribute
                           (luatexbase.attributes.g__tag_mc_cnt_attr))
                     }
                 }
                 {
                   \lua_now:e
                     {
                       ltx.__tag.trace.log
                        (
                          "mc-current:~no~MC~open,~current~abscnt
                           =\__tag_get_mc_abs_cnt:"
                          ,0
                        )
                       texio.write_nl("")
                     }
                 }
                 {
                   \lua_now:e
                     {
                       ltx.__tag.trace.log
                        (
                          "mc-current:~abscnt=\__tag_get_mc_abs_cnt:=="
                           ..
                           tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
                           ..
                           "~=>tag="
                           ..
                           tostring
                             (ltx.__tag.func.get_tag_from
                               (tex.getattribute
                                 (luatexbase.attributes.g__tag_mc_type_attr)))
                           ..
                           "="
                           ..
                           tex.getattribute
                            (luatexbase.attributes.g__tag_mc_type_attr)
                           ,0
                        )
                       texio.write_nl("")
                     }
                 }
             }
         }
         {
          \msg_note:nn{ tag }{ mc-current }
         }
     }
  }
```

(*End definition for* `mc-current`. *This function is documented on page* 25.)

`struct-stack`

```
\keys_define:nn { __tag / show }
  {
     struct-stack .choice:
    ,struct-stack / log .code:n = \seq_log:N \g__tag_struct_tag_stack_seq
    ,struct-stack / show .code:n = \seq_show:N \g__tag_struct_tag_stack_seq
    ,struct-stack .default:n = show
  }
```

(*End definition for* `struct-stack`. *This function is documented on page 25.*)

11 Commands to extend document commands

The following commands and code parts are not core command of tagpdf. The either provide work arounds for missing functionality elsewhere, or do a first step to apply tagpdf commands to document commands. This part should be regularly revisited to check if the code should go to a better place or can be improved.

11.1 Document structure

`\__tag_add_document_structure:n`
`activate`

```
\cs_new_protected:Npn \__tag_add_document_structure:n #1
 {
   \hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=#1}}
   \hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}
 }
\keys_define:nn { __tag / setup}
 {
   activate   .code:n =
    {
      \keys_set:nn { __tag / setup }
        { activate-mc,activate-tree,activate-struct }
      \__tag_add_document_structure:n {#1}
    },
  activate .default:n = Document
 }
```

(*End definition for* `\__tag_add_document_structure:n` *and* `activate`. *This function is documented on page* ??.)

11.2 Fake space

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

```
\sys_if_engine_luatex:T
  {
    \NewDocumentCommand\pdffakespace { }
     {
       \__tag_fakespace:
     }
  }
```

(*End definition for* `\pdffakespace`. *This function is documented on page 25.*)

11.3 Paratagging

The following are some simple commands to enable/disable paratagging. Probably one should add some checks if we are already in a paragraph.

\l__tag_para_bool
\l__tag_para_show_bool
\g__tag_para_int

At first some variables.

```
\bool_new:N \l__tag_para_bool
\bool_new:N \l__tag_para_show_bool
\int_new:N  \g__tag_para_int
```

(*End definition for* `\l__tag_para_bool`*,* `\l__tag_para_show_bool`*, and* `\g__tag_para_int`*.*)

paratagging
paratagging-show

These keys enable/disable locally paratagging, and the debug modus. It can affect the typesetting if `paratagging-show` is used. The small numbers are boxes and they have a (small) height.

```
\keys_define:nn { __tag / setup }
  {
    paratagging      .bool_set:N = \l__tag_para_bool,
    paratagging-show .bool_set:N = \l__tag_para_show_bool,
  }

```

(*End definition for* `paratagging` *and* `paratagging-show`*. These functions are documented on page 26.*)

This fills the para hooks with the needed code.

```
\AddToHook{para/begin}
  {
   \int_gincr:N \g__tag_para_int
   \bool_if:NT \l__tag_para_bool
     {
       \tag_struct_begin:n {tag=P}
       \bool_if:NT \l__tag_para_show_bool
        { \tag_mc_begin:n{artifact}
          \llap{\color_select:n{red}\tiny\int_use:N\g__tag_para_int\ }
          \tag_mc_end:
        }
       \tag_mc_begin:n {tag=P}
     }
  }
\AddToHook{para/end}
  {
    \bool_if:NT \l__tag_para_bool
      {
        \tag_mc_end:
        \bool_if:NT \l__tag_para_show_bool
          { \tag_mc_begin:n{artifact}
            \rlap{\color_select:n{red}\tiny\ \int_use:N\g__tag_para_int}
            \tag_mc_end:
          }
        \tag_struct_end:
      }
  }
```

\tagpdfparaOn
\tagpdfparaOff

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

```
\newcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
\newcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}
```

(*End definition for* `\tagpdfparaOn` *and* `\tagpdfparaOff`*. These functions are documented on page 26.*)

11.4 Links

We need to close and reopen mc-chunks around links. Currently we handle URI and GoTo (internal) links. Links should have an alternative text in the Contents key. It is unclear which text this should be and how to get it.

```
\hook_gput_code:nnn
  {pdfannot/link/URI/before}
  {tagpdf}
  {
    \tag_mc_end_push:
    \tag_struct_begin:n { tag=Link }
    \tag_mc_begin:n { tag=Link }
    \pdfannot_dict_put:nnx
      { link/URI }
      { StructParent }
      { \tag_struct_parent_int: }
  }

\hook_gput_code:nnn
  {pdfannot/link/URI/after}
  {tagpdf}
  {
     \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
     \tag_mc_end:
     \tag_struct_end:
     \tag_mc_begin_pop:n{}
  }

\hook_gput_code:nnn
  {pdfannot/link/GoTo/before}
  {tagpdf}
  {
     \tag_mc_end_push:
     \tag_struct_begin:n{tag=Link}
     \tag_mc_begin:n{tag=Link}
     \pdfannot_dict_put:nnx
       { link/GoTo }
       { StructParent }
       { \tag_struct_parent_int: }
  }

\hook_gput_code:nnn
  {pdfannot/link/GoTo/after}
  {tagpdf}
  {
    \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
    \tag_mc_end:
    \tag_struct_end:
    \tag_mc_begin_pop:n{}

  }

% "alternative descriptions " for PAX3. How to get better text here??
\pdfannot_dict_put:nnn
```

```
 { link/URI }
 { Contents }
 { (url) }

\pdfannot_dict_put:nnn
 { link/GoTo }
 { Contents }
 { (ref) }

```

</package>

Part III
The **tagpdf-tree** module Commands trees and main dictionaries Part of the tagpdf package

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-tree-code} {2021-07-03} {0.91}
 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
⟨/header⟩
```

1 Trees, pdfmanagement and finalization code

The code to finish the structure is in a hook. This will perhaps at the end be a kernel hook. TODO check right place for the code The pdfmanagement code is the kernel hook after shipout/lastpage so all code affecting it should be before. Objects can be written later, at least in pdf mode.

```
⟨*package⟩
\hook_gput_code:nnn{begindocument}{tagpdf}
  {
    \bool_if:NT \g__tag_active_tree_bool
      {
        \sys_if_output_pdf:TF
          {
            \AddToHook{enddocument/end} { \__tag_finish_structure: }
          }
          {
            \AddToHook{shipout/lastpage} { \__tag_finish_structure: }
          }
      }
  }
```

1.1 Catalog: MarkInfo and StructTreeRoot

The StructTreeRoot and the MarkInfo entry must be added to the catalog. We do it late so that we can win, but before the pdfmanagement hook.

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

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

(*End definition for* `__tag/struct/0`.)

```
\hook_gput_code:nnn{shipout/lastpage}{tagpdf}
  {
    \bool_if:NT \g__tag_active_tree_bool
      {
        \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
        \pdfmanagement_add:nnx
```

```
          { Catalog }
          { StructTreeRoot }
          { \pdf_object_ref:n { __tag/struct/0 } }
      }
  }
```

1.2 Writing structure elements

The following commands are needed to write out the structure.

__tag_tree_write_structtreeroot: This writes out the root object.

```
\cs_new_protected:Npn \__tag_tree_write_structtreeroot:
  {
     \__tag_prop_gput:cnx
       { g__tag_struct_0_prop }
       { ParentTree }
       { \pdf_object_ref:n { __tag/tree/parenttree } }
     \__tag_prop_gput:cnx
       { g__tag_struct_0_prop }
       { RoleMap }
       { \pdf_object_ref:n { __tag/tree/rolemap } }
     \__tag_struct_write_obj:n { 0 }
  }
```

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

__tag_tree_write_structelements: This writes out the other struct elems, the absolute number is in the counter

```
\cs_new_protected:Npn \__tag_tree_write_structelements:
  {
    \int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
      {
        \__tag_struct_write_obj:n { ##1 }
      }
  }
```

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

1.3 ParentTree

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

```
\pdf_object_new:nn { __tag/tree/parenttree }{ dict }
```

(*End definition for* `__tag/tree/parenttree`.)

The ParentTree maps numbers to objects or (if the number represents a page) to arrays of objects. The numbers refer to two dictinct types of entries: page streams and real objects like annotations. The numbers must be distinct and ordered. So we rely on abspage for the pages and put the real objects at the end. We use a counter to have a chance to get the correct number if code is processed twice.

\c@g__tag_parenttree_obj_int This is a counter for the real objects. It starts at the absolute last page value. It relies on l3ref.

```
\newcounter  { g__tag_parenttree_obj_int }
\hook_gput_code:nnn{begindocument}{tagpdf}
  {
    \int_gset:Nn
      \c@g__tag_parenttree_obj_int
      { \__tag_ref_value_lastpage:nn{abspage}{100}  }
  }
```

(*End definition for* `\c@g__tag_parenttree_obj_int`.)

We store the number/object references in a tl-var. If more structure is needed one could switch to a seq.

`\g__tag_parenttree_objr_tl`

```
\tl_new:N \g__tag_parenttree_objr_tl
```

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

`\__tag_parenttree_add_objr:nn` This command stores a StructParent number and a objref into the tl var. This is only for objects like annotations, pages are handled elsewhere.

```
\cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 %#1 StructParent number, #2 objref
  {
    \tl_gput_right:Nx \g__tag_parenttree_objr_tl
      {
        #1 \c_space_tl #2 ^^J
      }
  }
```

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

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

```
\tl_new:N \l__tag_parenttree_content_tl
```

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

`\__tag_tree_fill_parenttree:` This is the main command to assemble the page related entries of the parent tree. It wanders through the pages and the mcid numbers and collects all mcid of one page.

```

\cs_new_protected:Npn \__tag_tree_fill_parenttree:
  {
    \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{abspage}{-1}} %not quite clear i
      { %page ##1
        \prop_clear:N \l__tag_tmpa_prop
        \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{tagmcabs}{-1}}
          {
            %mcid####1
            \int_compare:nT
              {\__tag_ref_value:enn{mcid-####1}{tagabspage}{-1}=##1} %mcid is on current page
              {% yes
                \prop_put:Nxx
                  \l__tag_tmpa_prop
                  {\__tag_ref_value:enn{mcid-####1}{tagmcid}{-1}}
                  {\prop_item:Nn \g__tag_mc_parenttree_prop {####1}}
              }
          }
        \tl_put_right:Nx\l__tag_parenttree_content_tl
          {
            \int_eval:n {##1-1}\c_space_tl
            [\c_space_tl %]
          }
        \int_step_inline:nnnn
          {0}
          {1}
          { \prop_count:N \l__tag_tmpa_prop -1 }
          {
            \prop_get:NnNTF \l__tag_tmpa_prop {####1} \l__tag_tmpa_tl
              {% page#1:mcid##1:\l__tag_tmpa_tl :content
                \tl_put_right:Nx \l__tag_parenttree_content_tl
                  {
                    \pdf_object_if_exist:eT { __tag/struct/\l__tag_tmpa_tl }
                     {
                       \pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
                     }
                    \c_space_tl
                  }
              }
              {
                \msg_warning:nn { tag } {tree-mcid-index-wrong}
              }
          }
        \tl_put_right:Nn
          \l__tag_parenttree_content_tl
          {%[
            ]^^J
          }
      }
  }
```

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

`\__tag_tree_lua_fill_parenttree:` This is a special variant for luatex. lua mode must/can do it differently.

```
\cs_new_protected:Npn \__tag_tree_lua_fill_parenttree:
  {
    \tl_set:Nn \l__tag_parenttree_content_tl
      {
        \lua_now:e
          {
            ltx.__tag.func.output_parenttree
              (
                \int_use:N\g_shipout_readonly_int
              )
          }
      }
  }
```

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

`\__tag_tree_write_parenttree:` This combines the two parts and writes out the object. TODO should the check for lua be moved into the backend code?

```
\cs_new_protected:Npn \__tag_tree_write_parenttree:
  {
    \bool_if:NTF \g__tag_mode_lua_bool
      {
        \__tag_tree_lua_fill_parenttree:
      }
      {
        \__tag_tree_fill_parenttree:
      }
    \tl_put_right:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
    \pdf_object_write:nx  { __tag/tree/parenttree }
      {
        /Nums\c_space_tl [\l__tag_parenttree_content_tl]
      }
  }
```

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

1.4 Rolemap dictionary

The Rolemap dictionary describes relations between new tags and standard types. The main part here is handled in the role module, here we only define the command which writes it to the PDF.

`__tag/tree/rolemap` At first we reserve again an object.

```
\pdf_object_new:nn { __tag/tree/rolemap }{ dict }
```

(*End definition for* `__tag/tree/rolemap`.)

`\__tag_tree_write_rolemap:` This writes out the rolemap, basically it simply pushes out the dictionary which has been filled in the role module.

```
\cs_new_protected:Npn \__tag_tree_write_rolemap:
  {
    \pdf_object_write:nx  { __tag/tree/rolemap }
      {
        \pdfdict_use:n{g__tag_role/RoleMap_dict}
      }
  }
```

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

1.5 Classmap dictionary

Classmap and attributes are setup in the struct module, here is only the code to write it out. It should only done if values have been used.

`\__tag_tree_write_classmap:`

```
\cs_new_protected:Npn \__tag_tree_write_classmap:
  {
    \tl_clear:N \l__tag_tmpa_tl
    \seq_gremove_duplicates:N \g__tag_attr_class_used_seq
    \seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
      {
        /##1\c_space_tl
        <<
          \prop_item:Nn
            \g__tag_attr_entries_prop
            {##1}
        >>
      }
    \tl_set:Nx \l__tag_tmpa_tl
      {
        \seq_use:Nn
          \l__tag_tmpa_seq
          { \iow_newline: }
      }
    \tl_if_empty:NF
      \l__tag_tmpa_tl
      {
        \pdf_object_new:nn { __tag/tree/classmap }{ dict }
        \pdf_object_write:nx
          { __tag/tree/classmap }
          { \l__tag_tmpa_tl }
        \__tag_prop_gput:cnx
          { g__tag_struct_0_prop }
          { ClassMap }
          { \pdf_object_ref:n { __tag/tree/classmap }  }
      }
  }
```

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

1.6 Namespaces

Namespaces are handle in the role module, here is the code to write them out. Namespaces are only relevant for pdf2.0 but we don't care, it doesn't harm.

`__tag/tree/namespaces`

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

(*End definition for* `__tag/tree/namespaces`.)

`\__tag_tree_write_namespaces:`

```
\cs_new_protected:Npn \__tag_tree_write_namespaces:
 {
   \prop_map_inline:Nn \g__tag_role_NS_prop
     {
       \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
         {
           \pdf_object_write:nx {__tag/RoleMapNS/##1}
             {
               \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
             }
           \pdfdict_gput:nnx{g__tag_role/Namespace_##1_dict}
             {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
         }
       \pdf_object_write:nx{tag/NS/##1}
         {
            \pdfdict_use:n {g__tag_role/Namespace_##1_dict}
         }
     }
   \pdf_object_write:nx {__tag/tree/namespaces}
     {
       \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
     }
 }
```

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

1.7 Finishing the structure

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

`\__tag_finish_structure:`

```
\cs_new_protected:Npn \__tag_finish_structure:
  {
    \bool_if:NT\g__tag_active_tree_bool
      {
        \hook_use:n {tagpdf/finish/before}
        \__tag_tree_write_parenttree:
        \__tag_tree_write_rolemap:
        \__tag_tree_write_classmap:
        \__tag_tree_write_namespaces:
        \__tag_tree_write_structelements: %this is rather slow!!
        \__tag_tree_write_structtreeroot:
      }
  }
```

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

1.8 StructParents entry for Page

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

```
\hook_gput_code:nnn{begindocument}{tagpdf}
  {
    \bool_if:NT\g__tag_active_tree_bool
      {
       \hook_gput_code:nnn{shipout/before} { tagpdf/structparents }
         {
           \pdfmanagement_add:nnx
             { Page }
             { StructParents }
             { \int_eval:n { \g_shipout_readonly_int} }
         }
      }
  }
⟨/package⟩
```

Part IV
The **tagpdf-mc-shared** module Code related to Marked Content (mc-chunks), code shared by all modes Part of the tagpdf package

1 Public Commands

`\tag_mc_begin:n`
`\tag_mc_end:`

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

New: 2019-11-20

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

This command pair creates a group with an artifact marker at the begin and the end. Inside the group the tagging commands are disabled. It allows to mark a complete region as artifact without having to worry about user commands with tagging commands. ⟨*name*⟩ should be a value allowed also for the `artifact` key. It pushes and pops mc-chunks at the begin and end. TODO: document is in tagpdf.tex

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

New: 2021-04-22

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

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.

`\tag_mc_if_in_p:` ⋆
`\tag_mc_if_in:TF` ⋆

`\tag_mc_if_in:TF {⟨true code⟩} {⟨false code⟩}`

Determines if a mc-chunk is open.

2 Public keys

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

`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⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-mc-code-shared} {2021-07-03} {0.91}
  {part of tagpdf - code related to marking chunks -
   code shared by generic and luamode }
⟨/header⟩
```

3.1 Variables and counters

MC chunks must be counted. I use a latex counter for the absolute count, so that it is added to `\cl@@ckpt` and restored e.g. in tabulars and align. `\int_new:N  \c@g_@@_MCID_int` and `\tl_put_right:Nn\cl@@ckpt{\@elt{g_uf_test_int}}` would work too, but as the name is not expl3 then too, why bother? The absolute counter can be used to label and to check if the page counter needs a reset.

`g__tag_MCID_abs_int`

```
⟨*shared⟩
\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_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_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`
`\l__tag_mc_artifact_bool` This booleans store the stash and artifact status of the mc-chunk.

```
\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_set_label_used:n

Unlike with structures we can't check if a labeled mc has been used by looking at the P key, so we use a dedicated csname for the test

```
\cs_new_protected:Npn \__tag_mc_set_label_used:n #1 %#1 labelname
  {
    \tl_new:c { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
  }
```

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

\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 testing for struct the right test?

```
\cs_new_protected:Npn \tag_mc_use:n #1 %#1: label name
  {
    \__tag_check_if_active_struct:T
      {
        \tl_set:Nx  \l__tag_tmpa_tl { \__tag_ref_value:nnn{tagpdf-#1}{tagmcabs}{} }
        \tl_if_empty:NTF\l__tag_tmpa_tl
          {
            \msg_warning:nnn {tag} {mc-label-unknown} {#1}
          }
          {
            \cs_if_free:cTF { g__tag_mc_label_\tl_to_str:n{#1}_used_tl }
              {
                \__tag_mc_handle_stash:x { \l__tag_tmpa_tl }
```

```
                  \__tag_mc_set_label_used:n {#1}
                }
                {
                  \msg_warning:nnn {tag}{mc-used-twice}{#1}
                }
            }
        }
    }
```

(*End definition for* `\tag_mc_use:n`. *This function is documented on page 41.*)

`\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. It pushes and pops mc-chunks at the begin and end.

```
\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1
 {
  \tag_mc_end_push:
  \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:
  \tag_mc_begin_pop:n{}
 }
```

(*End definition for* `\tag_mc_artifact_group_begin:n` *and* `\tag_mc_artifact_group_end:`. *These functions are documented on page 41.*)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

```
\cs_new_protected:Npn \tag_mc_end_push:
  {
    \__tag_check_if_active_mc:T
      {
        \__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
  {
    \__tag_check_if_active_mc:T
      {
        \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 41.*)

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 63.*)

```
⟨/shared⟩
```

Part V

The **tagpdf-mc-generic** module Code related to Marked Content (mc-chunks), generic mode Part of the tagpdf package

1 Marked content code – generic mode

```
⟨@@=tag⟩
⟨*generic⟩
\ProvidesExplPackage {tagpdf-mc-code-generic} {2021-07-03} {0.91}
 {part of tagpdf - code related to marking chunks - generic mode}
⟨/generic⟩
```

1.1 Variables

`\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*⟩

```
⟨*generic⟩
\__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`.)

1.2 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 41.*)

__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 different.

```
\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`
`\__tag_mc_handle_stash:x`

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, it does *not* handle mc with the stash key .... TODO: why does luamode use it for begin + use, but generic mode only for begin?

```
\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 }
  }
\cs_generate_variant:Nn \__tag_mc_handle_stash:n { x }
```

(*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
  {
    \__tag_check_if_active_mc:T
      {
        \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:x { \int_use:N \c@g__tag_MCID_abs_int }
              }
          }
        \group_end:
      }
  }
\cs_new_protected:Nn \tag_mc_end:
  {
    \__tag_check_if_active_mc:T
      {
        \__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 41.*)

1.3 Keys

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

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

```
\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 62.*)

Part VI

The **tagpdf-mc-luacode** module Code related to Marked Content (mc-chunks), luamode-specific Part of the tagpdf package

The code is splitted into three parts: code shared by all engines, code specific to luamode and code not used by luamode.

1 Marked content code – luamode code

luamode uses attributes to mark mc-chunks. The two attributes used are defined in the backend file. The backend also load the lua file, as it can contain functions needed elsewhere. The attributes for mc are global (between 0.6 and 0.81 they were local but this was reverted). The attributes are setup only in lua, and one should use the lua functions to set and get them.
`g_@@_mc_type_attr`: the value represent the type
`g_@@_mc_cnt_attr`: will hold the `\c@g_@@_MCID_abs_int value`

Handling attribute needs a different system to number the page wise mcid's: a `\tagmcbegin ... \tagmcend` pair no longer surrounds exactly one mc chunk: it can be split at page breaks. We know the included mcid(s) only after the ship out. So for the `struct -> mcid` mapping we need to record `struct -> mc-cnt` (in `\g_@@_mc_parenttree_prop` and/or a lua table and at shipout `mc-cnt-> {mcid, mcid, ...}` and when building the trees connect both.

Key definitions are overwritten for luatex to store that data in lua-tables. The data for the mc are in `ltx.@@.mc[absnum]`. The fields of the table are:
tag : the type (a string)
raw : more properties (string)
label: a string.
artifact: the presence indicates an artifact, the value (string) is the type.
kids: a array of tables
`{1={kid=num2,page=pagenum1}, 2={kid=num2,page=pagenum2},...}`,
this describes the chunks the mc has been split to by the traversing code
parent: the number of the structure it is in. Needed to build the parent tree.

```
⟨@@=tag⟩
⟨*luamode⟩
\ProvidesExplPackage {tagpdf-mc-code-lua} {2021-07-03} {0.91}
  {tagpdf - mc code only for the luamode }
⟨/luamode⟩
```

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

```
⟨*luamode⟩
\hook_gput_code:nnn{begindocument}{tagpdf/mc}
  {
    \bool_if:NT\g__tag_active_space_bool
      {
        \lua_now:e
          {
           if~luatexbase.callbacktypes.pre_shipout_filter~then~
              luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)~
              ltx.__tag.func.space_chars_shipout(TAGBOX)~return~true~
              end, "tagpdf")~
            end
          }
       \lua_now:e
         {
           if~luatexbase.callbacktypes.pre_shipout_filter~then~
           token.get_next()~
           end
         }\@secondoftwo\@gobble
           {
             \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
               {
                \lua_now:e
                   { ltx.__tag.func.space_chars_shipout (tex.box["ShipoutBox"]) }
               }
           }
      }
    \bool_if:NT\g__tag_active_mc_bool
      {
        \lua_now:e
          {
            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
          }
       \lua_now:e
         {
           if~luatexbase.callbacktypes.pre_shipout_filter~then~
           token.get_next()~
           end
         }\@secondoftwo\@gobble
           {
             \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
               {
                 \lua_now:e
                   { ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
               }
           }
      }
  }
```

1.1 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 }
      =
      {\lua_now:e
         {
           tex.print(tex.getattribute(luatexbase.attributes.g__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
  {
    %TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
    \tl_set:Nx\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from ("#1")} }
    \lua_now:e
      {
        tex.setattribute
         (
          "global",
          luatexbase.attributes.g__tag_mc_type_attr,
          \l__tag_tmpa_tl
         )
      }
    \lua_now:e
      {
        tex.setattribute
         (
           "global",
           luatexbase.attributes.g__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:
  {
    \lua_now:e
      {
        tex.setattribute
          (
            "global",
            luatexbase.attributes.g__tag_mc_type_attr,
            -2147483647
          )
      }
     \lua_now:e
      {
        tex.setattribute
          (
            "global",
            luatexbase.attributes.g__tag_mc_cnt_attr,
            -2147483647
          )
      }
  }

```

(*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
  {
    \lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
  }

\cs_new:Nn \__tag_mc_insert_mcid_single_kids:n
  {
    \lua_now:e {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:x`

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}%
      }
    \lua_now:e
      {
        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 { x }
```

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

`\tag_mc_begin:n` This is the lua version of the user command. We currently don't check if there is nesting as it doesn't matter so much in lua.

```
\cs_new_protected:Nn \tag_mc_begin:n
  {
    \__tag_check_if_active_mc:T
      {
        \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
              {
                \__tag_mc_handle_stash:x { \__tag_get_mc_abs_cnt: }
              }
          }
        \group_end:
     }
  }
```

(*End definition for* `\tag_mc_begin:n`*. This function is documented on page 41.*)

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

```
\cs_new_protected:Nn \tag_mc_end:
  {
    \__tag_check_if_active_mc:T
      {
        %\__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 { }
        \tl_gset:Nn \g__tag_mc_key_tag_tl { }
      }
  }
```

(*End definition for* `\tag_mc_end:`*. This function is documented on page 41.*)

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

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

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

1.2 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 = %
      {
        \tl_set:Nx   \l__tag_mc_key_tag_tl { #1 }
        \tl_gset:Nx  \g__tag_mc_key_tag_tl { #1 }
        \lua_now:e
          {
            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 }
        \lua_now:e
          {
            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>~ }
        \lua_now:e
          {
            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>~ }
        \lua_now:e
          {
            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>~ }
        \lua_now:e
          {
            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>~ }
        \lua_now:e
          {
            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 }
        \lua_now:e
          {
            ltx.__tag.func.store_mc_data
              (
                \__tag_get_mc_abs_cnt:,"label","#1"
              )
          }
      },
    __artifact-store .code:n =
      {
        \lua_now:e
          {
            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 62.*)

Part VII

The **tagpdf-struct** module Commands to create the structure Part of the tagpdf package

1 Public Commands

`\tag_struct_begin:n` `\tag_struct_end:`

```
\tag_struct_begin:n{⟨key-values⟩}
\tag_struct_end:
```

These commands start and end a new structure. They don't start a group. They set all their values globally.

`\tag_struct_use:n`

```
\tag_struct_use:n{⟨label⟩}
```

These commands insert a structure previously stashed away as kid into the currently active structure. A structure should be used only once, if the structure already has a parent a warning is issued.

The following two functions are used to add annotations. They must be used together and with care to get the same numbers. Perhaps some improvements are needed here.

`\tag_struct_insert_annot:nn`

```
\tag_struct_insert_annot:nn{⟨object reference⟩}{⟨struct parent number⟩}
```

This inserts an annotation in the structure. ⟨*object reference*⟩ is there reference to the annotation. ⟨*struct parent number*⟩ should be the same number as had been inserted with `\tag_struct_parent_int:` as `StructParent` value to the dictionary of the annotion. The command will increase the value of the counter used by `\tag_struct_parent_int:`.

`\tag_struct_parent_int:`

```
\tag_struct_parent_int:
```

This gives back the next free /StructParent number (assuming that it is together with `\tag_struct_insert_annot:nn` which will increase the number.

2 Public keys

2.1 Keys for the structure commands

`tag` This is required. The value of the key is normally one of the standard types listed in section ??. It is possible to setup new tags/types. The value can also be of the form `type/NS`, where `NS` is the shorthand of a declared name space. Currently the names spaces `pdf`, `pdf2`, `mathml` and `user` are defined. This allows to use a different name space than the one connected by default to the tag. But normally this should not be needed.

`stash` Normally a new structure inserts itself as a kid into the currently active structure. This key prohibits this. The structure is nevertheless from now on "the current active structure" and parent for following marked content and structures.

`label` This key sets a label by which one can use the structure later in another structure. Internally the label name will start with `tagpdfstruct-`.

`title`
`title-o` This keys allows to set the dictionary entry `/Title` in the structure object. The value is handled as verbatim string and hex encoded. Commands are not expanded. `title-o` will expand the value once.

`alttext`
`alttext-o` This key inserts an `/Alt` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. `alttext-o` will expand the value once.

`actualtext`
`actualtext-o` This key inserts an `/ActualText` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. `actualtext-o` will expand the value once.

`lang` This key allows to set the language for a structure element. The value should be a bcp-identifier, e.g. `de-De`.

`ref` This key allows to add references to other structure elements, it adds the `/Ref` array to the structure. The value should be a comma separated list of structure labels set with the `label` key. e.g. `ref={label1,label2}`.

`E` This key sets the `/E` key, the expanded form of an abbreviation or an acronym (I couldn't think of a better name, so I sticked to E).

`AF`
`AFinline`
`AFinline-o`

`AF = ⟨object name⟩`
`AF-inline = ⟨text content⟩`

These keys allows to reference an associated file in the structure element. The value ⟨*object name*⟩ should be the name of an object pointing to the `/Filespec` dictionary as expected by `\pdf_object_ref:n` from a current `l3kernel`.

The value `AF-inline` is some text, which is embedded in the PDF as a text file with mime type text/plain. `AF-inline-o` is like `AF-inline` but expands the value once.

Future versions will perhaps extend this to more mime types, but it is still a research task to find out what is really needed.

`attribute` This key takes as argument a comma list of attribute names (use braces to protect the commas from the external key-val parser) and allows to add one or more attribute dictionary entries in the structure object. As an example

```
\tagstructbegin{tag=TH,attribute= TH-row}
```

Attribute names and their content must be declared first in `\tagpdfsetup`.

`attribute-class` This key takes as argument a comma list of attribute class names (use braces to protect the commas from the external key-val parser) and allows to add one or more attribute classes to the structure object.

Attribute class names and their content must be declared first in `\tagpdfsetup`.

2.2 Setup keys

`newattribute` `newattribute = {⟨name⟩}{⟨Content⟩}`

This key can be used in the setup command `\tagpdfsetup` and allow to declare a new attribute, which can be used as attribute or attribute class. The value are two brace groups, the first contains the name, the second the content.

```
\tagpdfsetup
 {
  newattribute =
   {TH-col}{/O /Table /Scope /Column},
  newattribute =
   {TH-row}{/O /Table /Scope /Row},
  }
```

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-struct-code} {2021-07-03} {0.91}
 {part of tagpdf - code related to storing structure}
⟨/header⟩
```

3 Variables

`\c@g__tag_struct_abs_int` Every structure will have a unique, absolute number. I will use a latex counter for the structure count to have a chance to avoid double structures in align etc.

```
⟨*package⟩
\newcounter  { g__tag_struct_abs_int }
\int_gzero:N \c@g__tag_struct_abs_int
```

(*End definition for* `\c@g__tag_struct_abs_int`.)

`\g__tag_struct_objR_seq` a sequence to store mapping between the structure number and the object number. We assume that structure numbers are assign consecutively and so the index of the seq can be used. A seq allows easy mapping over the structures.

```
\__tag_seq_new:N  \g__tag_struct_objR_seq
```

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

`\g__tag_struct_stack_seq` A stack sequence for the structure stack. When a sequence is opened it's number is put on the stack.

```
\seq_new:N    \g__tag_struct_stack_seq
\seq_gpush:Nn \g__tag_struct_stack_seq {0}
```

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

`\g__tag_struct_tag_stack_seq` We will perhaps also need the tags. While it is possible to get them from the numbered stack, lets build a tag stack too.

```
\seq_new:N    \g__tag_struct_tag_stack_seq
\seq_gpush:Nn \g__tag_struct_tag_stack_seq {Root}
```

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

`\g__tag_struct_stack_current_tl`
`\l__tag_struct_stack_parent_tmpa_tl` The global variable will hold the current structure number. The local temporary variable will hold the parent when we fetch it from the stack.

```
\tl_new:N     \g__tag_struct_stack_current_tl
\tl_new:N     \l__tag_struct_stack_parent_tmpa_tl
```

(*End definition for* `\g__tag_struct_stack_current_tl` *and* `\l__tag_struct_stack_parent_tmpa_tl`.)

I will need at least one structure: the StructTreeRoot normally it should have only one kid, e.g. the document element.

The data of the StructTreeRoot and the StructElem are in properties: `\g_@@_struct_0_prop` for the root and `\g_@@_struct_N_prop`, $N \geq = 1$ for the other.

This creates quite a number of properties, so perhaps we will have to do this more efficiently in the future.

All properties have at least the keys

Type StructTreeRoot or StructElem

and the keys from the two following lists (the root has a special set of properties). the values of the prop should be already escaped properly when the entries are created (title,lange,alt,E,actualtext)

`\c__tag_struct_StructTreeRoot_entries_seq`
`\c__tag_struct_StructElem_entries_seq` These seq contain the keys we support in the two object types. They are currently no longer used, but are provided as documentation and for potential future checks. They should be adapted if there are changes in the PDF format.

```
\seq_const_from_clist:Nn \c__tag_struct_StructTreeRoot_entries_seq
  {%p. 857/858
    Type,              % always /StructTreeRoot
    K,                 % kid, dictionary or array of dictionaries
    IDTree,            % currently unused
    ParentTree,        % required,obj ref to the parent tree
    ParentTreeNextKey, % optional
    RoleMap,
    ClassMap,
    Namespaces
  }

\seq_const_from_clist:Nn \c__tag_struct_StructElem_entries_seq
  {%p 858 f
    Type,              %always /StructElem
    S,                  %tag/type
    P,                  %parent
    ID,                 %optional
    Ref,                %optional, pdf 2.0 Use?
    Pg,                 %obj num of starting page, optional
    K,                  %kids
    A,                  %attributes, probably unused
    C,                  %class ""
    %R,                 %attribute revision number, irrelevant for us as we
                        % don't update/change existing PDF and (probably)
                        % deprecated in PDF 2.0
    T,                  %title, value in () or <>
    Lang,               %language
    Alt,                % value in () or <>
    E,                  % abreviation
    ActualText,
    AF,                  %pdf 2.0, array of dict, associated files
    NS,                  %pdf 2.0, dict, namespace
    PhoneticAlphabet,    %pdf 2.0
    Phoneme              %pdf 2.0
  }
```

(*End definition for* `\c__tag_struct_StructTreeRoot_entries_seq` *and* `\c__tag_struct_StructElem_-entries_seq`.)

3.1 Variables used by the keys

`\g__tag_struct_tag_tl`
`\g__tag_struct_tag_NS_tl`

Use by the tag key to store the tag and the namespace.

```
\tl_new:N \g__tag_struct_tag_tl
\tl_new:N \g__tag_struct_tag_NS_tl
```

(*End definition for* `\g__tag_struct_tag_tl` *and* `\g__tag_struct_tag_NS_tl`.)

`\l__tag_struct_key_label_tl`

This will hold the label value.

```
\tl_new:N \l__tag_struct_key_label_tl
```

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

`\l__tag_struct_elem_stash_bool`

This will keep track of the stash status

```
\bool_new:N \l__tag_struct_elem_stash_bool
```

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

4 Commands

The properties must be in some places handled expandably. So I need an output handler for each prop, to get expandable output see https://tex.stackexchange.com/questions/424208. There is probably room here for a more efficient implementation. TODO check if this can now be implemented with the pdfdict commands. The property contains currently non pdf keys, but e.g. object numbers are perhaps no longer needed as we have named object anyway.

__tag_struct_output_prop_aux:nn
__tag_new_output_prop_handler:n

```
\cs_new:Npn \__tag_struct_output_prop_aux:nn #1 #2 %#1 num, #2 key
  {
    \prop_if_in:cnT
      { g__tag_struct_#1_prop }
      { #2 }
      {
        \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 }
      }
  }

\cs_new_protected:Npn \__tag_new_output_prop_handler:n #1
  {
    \cs_new:cn { __tag_struct_output_prop_#1:n }
      {
        \__tag_struct_output_prop_aux:nn {#1}{##1}
      }
  }
```

(*End definition for* `\__tag_struct_output_prop_aux:nn` *and* `\__tag_new_output_prop_handler:n`.)

4.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object is `@@/struct/0` which is currently created in the tree code (TODO move it here). The `ParentTree` and `RoleMap` entries are added at begin document in the tree code as they refer to object which are setup in other parts of the code. This avoid timing issues.

```
\tl_gset:Nn \g__tag_struct_stack_current_tl {0}
```

g__tag_struct_0_prop
g__tag_struct_kids_0_seq

```
\__tag_prop_new:c { g__tag_struct_0_prop }
\__tag_new_output_prop_handler:n {0}
\__tag_seq_new:c  { g__tag_struct_kids_0_seq }

\__tag_prop_gput:cnn
  { g__tag_struct_0_prop }
  { Type }
  { /StructTreeRoot }



```

Namespaces are pdf 2.0 but it doesn't harm to have an empty entry. We could add a test, but if the code moves into the kernel, timing could get tricky.

```
\__tag_prop_gput:cnx
  { g__tag_struct_0_prop }
  { Namespaces }
  { \pdf_object_ref:n { __tag/tree/namespaces } }
```

(*End definition for* `g__tag_struct_0_prop` *and* `g__tag_struct_kids_0_seq`.)

4.2 Handlings kids

Commands to store the kids. Kids in a structure can be a reference to a mc-chunk, an object reference to another structure element, or a object reference to an annotation (through an OBJR object).

__tag_struct_kid_mc_gput_right:nn

The command to store an mc-chunk, this is a dictionary of type MCR. It would be possible to write out the content directly as unnamed object and to store only the object reference, but probably this would be slower, and the PDF is more readable like this. The code doesn't try to avoid the use of the /Pg key by checking page numbers. That imho only slows down without much gain.

```
\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 MCID absn
  {
    \__tag_seq_gput_right:cx
      { g__tag_struct_kids_#1_seq }
      {
        <<
        /Type \c_space_tl /MCR \c_space_tl
        /Pg
          \c_space_tl
          \pdf_pageobject_ref:n { \__tag_ref_value:enn{mcid-#2}{tagabspage}{1} }
        /MCID \c_space_tl \__tag_ref_value:enn{mcid-#2}{tagmcid}{1}
        >>
      }
  }
```

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

__tag_struct_kid_struct_gput_right:nn
__tag_struct_kid_struct_gput_right:xx

This commands adds a structure as kid. We only need to record the object reference in the sequence.

```
\cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent struct, #2
  {
    \__tag_seq_gput_right:cx
      { g__tag_struct_kids_#1_seq }
      {
        \pdf_object_ref:n { __tag/struct/#2 }
      }
 }

\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {xx}
```

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

__tag_struct_kid_OBJR_gput_right:nn
__tag_struct_kid_OBJR_gput_right:xx

At last the command to add an OBJR object. This has to write an object first. The first argument is the number of the parent structure, the second the (expanded) object reference of the annotation.

```
\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nn #1 #2 %#1 num of parent struct,
                                                                %#2 obj reference
  {
    \pdf_object_unnamed_write:nn
      { dict }
      {
        /Type/OBJR/Obj~#2
      }
    \__tag_seq_gput_right:cx
      { g__tag_struct_kids_#1_seq }
      {
        \pdf_object_ref_last:
      }
  }

\cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nn { xx }

```

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

`\__tag_struct_exchange_kid_command:N`
`\__tag_struct_exchange_kid_command:c`

In luamode it can happen that a single kid in a structure is split at a page break into two or more mcid. In this case the lua code has to convert put the dictionary of the kid into an array. See issue 13 at tagpdf repo. We exchange the dummy command for the kids to mark this case.

```
\cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
  {
    \seq_gpop_left:NN #1 \l__tag_tmpa_tl
    \regex_replace_once:nnN
      { \c{\__tag_mc_insert_mcid_kids:n} }
      { \c{\__tag_mc_insert_mcid_single_kids:n} }
      \l__tag_tmpa_tl
    \seq_gput_left:NV #1 \l__tag_tmpa_tl
  }

\cs_generate_variant:Nn\__tag_struct_exchange_kid_command:N { c }
```

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

`\__tag_struct_fill_kid_key:n`

This command adds the kid info to the K entry. In lua mode the content contains commands which are expanded later. The argument is the structure number.

```
\cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1 %#1 is the struct num
  {
    \int_case:nnF
      {
        \seq_count:c
          {
            g__tag_struct_kids_#1_seq
          }
      }
      {
        { 0 }
         { } %no kids, do nothing
        { 1 } % 1 kid, insert
         {
           % in this case we need a special command in
           % luamode to get the array right. See issue #13
           \bool_if:NT\g__tag_mode_lua_bool
             {
               \__tag_struct_exchange_kid_command:c
                {g__tag_struct_kids_#1_seq}
             }
           \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
```

```
                {
                  \seq_item:cn
                    {
                      g__tag_struct_kids_#1_seq
                    }
                    {1}
                }
            } %
        }
        { %many kids, use an array
          \__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
            {
              [
                \seq_use:cn
                  {
                    g__tag_struct_kids_#1_seq
                  }
                  {
                    \c_space_tl
                  }
              ]
            }
        }
  }

```

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

`\__tag_struct_get_dict_content:nN` This maps the dictionary content of a structure into a tl-var. Basically it does what `\pdfdict_use:n` does. TODO!! this looks over-complicated. Check if it can be done with pdfdict now.

```
\cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: stucture num
  {
    \tl_clear:N #2
    \seq_map_inline:cn
      {
        c__tag_struct_
         \int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
         _entries_seq
      }
      {
        \tl_put_right:Nx
          #2
          {
              \prop_if_in:cnT
                { g__tag_struct_#1_prop }
                { ##1 }
                {
                  \c_space_tl/##1~\prop_item:cn{ g__tag_struct_#1_prop } { ##1 }
                }
          }
      }
  }
```

(*End definition for* `\__tag_struct_get_dict_content:nN`.)

__tag_struct_write_obj:n This writes out the structure object. This is done in the finish code, in the tree module and guarded by the tree boolean.

```
\cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
  {
    \pdf_object_if_exist:nTF { __tag/struct/#1 }
      {
        \__tag_struct_fill_kid_key:n { #1 }
        \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
        \exp_args:Nx
          \pdf_object_write:nx
            { __tag/struct/#1 }
            {
              \l__tag_tmpa_tl
            }
      }
      {
        \msg_error:nnn { tag } { struct-no-objnum } { #1}
      }
  }
```

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

__tag_struct_insert_annot:nn This is the command to insert an annotation into the structure. It can probably be used for xform too.

Annotations used as structure content must

1. add a StructParent integer to their dictionary
2. push the object reference as OBJR object in the structure
3. Add a Structparent/obj-nr reference to the parent tree.

For a link this looks like this

```
        \tag_struct_begin:n { tag=Link }
        \tag_mc_begin:n { tag=Link }
(1)     \pdfannot_dict_put:nnx
          { link/URI }
          { StructParent }
          { \int_use:N\c@g_@@_parenttree_obj_int }
   <start link> link text <stop link>
(2+3)   \@@_struct_insert_annot:nn {obj ref}{parent num}
        \tag_mc_end:
        \tag_struct_end:
```

```
\cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotati
                                                          %#2 structparent number
  {
    \bool_if:NT \g__tag_active_struct_bool
      {
        %get the number of the parent structure:
        \seq_get:NNF
          \g__tag_struct_stack_seq
          \l__tag_struct_stack_parent_tmpa_tl
          {
            \msg_error:nn { tag } { struct-faulty-nesting }
          }
        %put the obj number of the annot in the kid entry, this also creates
        %the OBJR object
        \__tag_struct_kid_OBJR_gput_right:xx
          {
            \l__tag_struct_stack_parent_tmpa_tl
          }
          {
            #1 %
          }
        % add the parent obj number to the parent tree:
        \exp_args:Nnx
        \__tag_parenttree_add_objr:nn
          {
            #2
          }
          {
            \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
          }
        % increase the int:
        \stepcounter{ g__tag_parenttree_obj_int }
      }
  }
```

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

`\__tag_get_data_struct_tag:`

this command allows `\tag_get:n` to get the current structure tag with the keyword `struct_tag`. We will need to handle nesting

```
\cs_new:Npn \__tag_get_data_struct_tag:
  {
    \exp_args:Ne
    \tl_tail:n
     {
       \prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl _prop}{S}
     }
  }
```

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

5 Keys

This are the keys for the user commands. we store the tag in a variable. But we should be careful, it is only reliable at the begin.

`label`
`stash`
`tag`
`title`
`title-o`
`alttext`
`alttext-o`
`actualtext`
`actualtext-o␣␣␣␣lang`
`ref`
`E`

```
\keys_define:nn { __tag / struct }
  {
    label .tl_set:N      = \l__tag_struct_key_label_tl,
    stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
    tag   .code:n        = % S property
      {
        \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Nn\g__tag_role_tags_NS_prop{
```

```
        \tl_gset:Nx \g__tag_struct_tag_tl    { \seq_item:Nn\l__tag_tmpa_seq {1} }
        \tl_gset:Nx \g__tag_struct_tag_NS_tl { \seq_item:Nn\l__tag_tmpa_seq {2} }
        \__tag_check_structure_tag:N \g__tag_struct_tag_tl
        \__tag_prop_gput:cnx
         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
         { S }
         { \pdf_name_from_unicode_e:n{ \g__tag_struct_tag_tl} } %
       \prop_get:NVNT \g__tag_role_NS_prop\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl
         {
           \__tag_prop_gput:cnx
            { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
            { NS }
            { \l__tag_tmpa_tl } %
         }
      },
    title .code:n        = % T property
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { T }
          { <\l__tag_tmpa_str> }
      },
    title-o .code:n        = % T property
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { T }
          { <\l__tag_tmpa_str> }
      },
    alttext .code:n      = % Alt property
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { Alt }
          { <\l__tag_tmpa_str> }
      },
    alttext-o .code:n      = % Alt property
      {
        \str_set_convert:Noon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { Alt }
          { <\l__tag_tmpa_str> }
      },
    actualtext .code:n  = % ActualText property
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { ActualText }
          { <\l__tag_tmpa_str>}
      },
    actualtext-o .code:n  = % ActualText property
      {
        \str_set_convert:Noon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { ActualText }
          { <\l__tag_tmpa_str>}
      },
    lang .code:n        = % Lang property
      {
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { Lang }
          { (#1) }
      },
    ref .code:n        = % Lang property
      {
        \tl_clear:N\l__tag_tmpa_tl
        \clist_map_inline:nn {#1}
          {
            \tl_put_right:Nx \l__tag_tmpa_tl
              {~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} }
          }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { Ref }
          { [\l__tag_tmpa_tl] }
      },
    E .code:n        = % E property
```

```
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { E }
          { <\l__tag_tmpa_str> }
      },
  }
```

(*End definition for* `label` *and others. These functions are documented on page 63.*)

AF
AFinline
AFinline-o

keys for the AF keys (associated files). They use commands from l3pdffile! The stream variants use txt as extension to get the mimetype. TODO: check if this should be configurable. For math we will perhaps need another extension.

```
\keys_define:nn { __tag / struct }
 {
    AF .code:n        = % AF property
      {
        \pdf_object_if_exist:nTF {#1}
          {
            \__tag_prop_gput:cnx
             { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
             { AF }
             { \pdf_object_ref:n {#1} }
          }
          {

          }
      },
   ,AFinline .code:n =
     {
       \group_begin:
       \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
        {
          \pdffile_embed_stream:nxx
            {#1}
            {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
            {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
        }
       \__tag_prop_gput:cnx
         { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
         { AF }
         { \pdf_object_ref:e {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int } }
       \group_end:
     }
   ,AFinline-o .code:n =
     {
       \group_begin:
       \pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
        {
          \pdffile_embed_stream:oxx
            {#1}
            {tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
            {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
        }
       \__tag_prop_gput:cnx
         { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
         { AF }
         { \pdf_object_ref:e {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int } }
       \group_end:
     }
}
```

(*End definition for* `AF`*,* `AFinline`*, and* `AFinline-o`*. These functions are documented on page 63.*)

6 User commands

`\tag_struct_begin:n`
`\tag_struct_end:`

```
\cs_new_protected:Npn \tag_struct_begin:n #1 %#1 key-val
  {
    \__tag_check_if_active_struct:T
      {
        \group_begin:
        \int_gincr:N \c@g__tag_struct_abs_int
        \__tag_prop_new:c  { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
        \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
        \__tag_seq_new:c  { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
        \exp_args:Ne
          \pdf_object_new:nn
            { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
            { dict }
        \__tag_prop_gput:cno
          { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
          { Type }
          { /StructElem }
        \keys_set:nn { __tag / struct} { #1 }
        \__tag_check_structure_has_tag:n { \int_eval:n {\c@g__tag_struct_abs_int} }
        \tl_if_empty:NF
          \l__tag_struct_key_label_tl
          {
            \__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_tl}{struct}
          }
        %get the potential parent from the stack:
        \seq_get:NNF
          \g__tag_struct_stack_seq
          \l__tag_struct_stack_parent_tmpa_tl
          {
            \msg_error:nn { tag } { struct-faulty-nesting }
          }
        \seq_gpush:NV \g__tag_struct_stack_seq        \c@g__tag_struct_abs_int
        \seq_gpush:NV \g__tag_struct_tag_stack_seq    \g__tag_struct_tag_tl
        \tl_gset:NV   \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
        %\seq_show:N   \g__tag_struct_stack_seq
        \bool_if:NF
          \l__tag_struct_elem_stash_bool
          {%set the  parent
            \__tag_prop_gput:cnx
              { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
              { P }
              {
                \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
              }
            %record this structure as kid:
            %\tl_show:N \g__tag_struct_stack_current_tl
            %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
            \__tag_struct_kid_struct_gput_right:xx
               { \l__tag_struct_stack_parent_tmpa_tl }
               { \g__tag_struct_stack_current_tl }
            %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
            %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
          }
        %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
        %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
        \group_end:
     }
  }


\cs_new_protected:Nn \tag_struct_end:
  { %take the current structure num from the stack:
    %the objects are written later, lua mode hasn't all needed info yet
    %\seq_show:N \g__tag_struct_stack_seq
    \__tag_check_if_active_struct:T
      {
        \seq_gpop:NN   \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
        \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
          {
            \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
          }
          { \__tag_check_no_open_struct: }
        % get the previous one, shouldn't be empty as the root should be there
        \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
          {
            \tl_gset:NV   \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
          }
          {
            \__tag_check_no_open_struct:
          }
       \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
          {
            \tl_gset:NV \g__tag_struct_tag_tl \l__tag_tmpa_tl
          }
      }
  }
```

(*End definition for* `\tag_struct_begin:n` *and* `\tag_struct_end:`. *These functions are documented on page 62.*)

`\tag_struct_use:n` This command allows to use a stashed structure in another place. TODO: decide how it should be guarded. Probably by the struct-check.

```
\cs_new_protected:Nn \tag_struct_use:n %#1 is the label
  {
    \__tag_check_if_active_struct:T
      {
        \prop_if_exist:cTF
          { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
          {
            \__tag_check_struct_used:n {#1}
            %add the label structure as kid to the current structure (can be the root)
            \__tag_struct_kid_struct_gput_right:xx
              { \g__tag_struct_stack_current_tl }
              { \__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0} }
            %add the current structure to the labeled one as parents
            \__tag_prop_gput:cnx
              { g__tag_struct_\__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop }
              { P }
              {
                \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl }
              }
          }
          {
            \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
          }
      }
  }
```

(*End definition for* `\tag_struct_use:n`. *This function is documented on page 62.*)

`\tag_struct_insert_annot:nn`
`\tag_struct_insert_annot:xx`
`\tag_struct_parent_int:`

This are the user command to insert annotations. They must be used together to get the numbers right. They use a counter to the `StructParent` and `\tag_struct_insert_annot:nn` increases the counter given back by `\tag_struct_parent_int:`.

It must be used together with `\tag_struct_parent_int:` to insert an annotation. TODO: decide how it should be guarded if tagging is deactivated.

```
\cs_new_protected:Npn \tag_struct_insert_annot:nn #1 #2 %#1 should be an object reference
                                                         %#2 struct parent num
  {
    \__tag_check_if_active_struct:T
      {
        \__tag_struct_insert_annot:nn {#1}{#2}
      }
  }

\cs_generate_variant:Nn \tag_struct_insert_annot:nn {xx}
\cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}

⟨/package⟩

```

(*End definition for* `\tag_struct_insert_annot:nn` *and* `\tag_struct_parent_int:`. *These functions are documented on page 62.*)

7 Attributes and attribute classes

```
⟨*header⟩
\ProvidesExplPackage {tagpdf-attr-code} {2021-07-03} {0.91}
  {part of tagpdf - code related to attributes and attribute classes}
⟨/header⟩
```

7.1 Variables

\g__tag_attr_entries_prop
\g__tag_attr_class_used_seq
\g__tag_attr_objref_prop
\l__tag_attr_value_tl

`\g_@@_attr_entries_prop` will store attribute names and their dictionary content. `\g_@@_attr_class_used_seq` will hold the attributes which have been used as class name. `\l_@@_attr_value_tl` is used to build the attribute array or key. Everytime an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in `\g_@@_attr_objref_prop`

```
⟨*package⟩
\prop_new:N \g__tag_attr_entries_prop
\seq_new:N  \g__tag_attr_class_used_seq
\tl_new:N   \l__tag_attr_value_tl
\prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes
```

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

7.2 Commands and keys

__tag_attr_new_entry:nn
newattribute

This allows to define attributes. Defined attributes are stored in a global property. `newattribute` expects two brace group, the name and the content. The content typically needs an `/O` key for the owner. An example look like this.

```
\tagpdfsetup
 {
  newattribute =
   {TH-col}{/O /Table /Scope /Column},
  newattribute =
   {TH-row}{/O /Table /Scope /Row},
  }
```

```
\cs_new_protected:Npn \__tag_attr_new_entry:nn #1 #2 %#1:name, #2: content
  {
    \prop_gput:Nnn \g__tag_attr_entries_prop
      {#1}{#2}
  }

\keys_define:nn { __tag / setup }
  {
    newattribute .code:n =
      {
        \__tag_attr_new_entry:nn #1
      }
  }
```

(*End definition for* `\__tag_attr_new_entry:nn` *and* `newattribute`. *This function is documented on page 64.*)

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

```
\keys_define:nn { __tag / struct }
  {
    attribute-class .code:n =
     {
       \clist_set:No \l__tag_tmpa_clist { #1 }
       \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
       \seq_map_inline:Nn \l__tag_tmpa_seq
         {
           \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
             {
               \msg_error:nnn { tag } { attr-unknown } { ##1 }
             }
           \seq_gput_left:Nn\g__tag_attr_class_used_seq { ##1}
         }
       \seq_set_map:NNn \l__tag_tmpb_seq \l__tag_tmpa_seq
         {
           /##1
         }
       \tl_set:Nx \l__tag_tmpa_tl
         {
           \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}
           \seq_use:Nn \l__tag_tmpb_seq  { \c_space_tl  }
           \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}
         }
       \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 0 }
         {
           \__tag_prop_gput:cnx
             { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
             { C }
             { \l__tag_tmpa_tl }
          %\prop_show:c  { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
         }
     }
  }
```

(*End definition for* `attribute-class`. *This function is documented on page 64.*)

attribute

```
\keys_define:nn { __tag / struct }
  {
    attribute .code:n  = % A property (attribute, value currently a dictionary)
      {
        \clist_set:No          \l__tag_tmpa_clist { #1 }
        \seq_set_from_clist:NN \l__tag_tmpa_seq \l__tag_tmpa_clist
        \tl_set:Nx \l__tag_attr_value_tl
          {
            \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}%]
          }
        \seq_map_inline:Nn \l__tag_tmpa_seq
          {
            \prop_if_in:NnF \g__tag_attr_entries_prop {##1}
              {
                \msg_error:nnn { tag } { attr-unknown } { ##1 }
              }
            \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
              {%\prop_show:N \g__tag_attr_entries_prop
                \pdf_object_unnamed_write:nx
                  { dict }
                  {
                    \prop_item:Nn\g__tag_attr_entries_prop {##1}
                  }
                \prop_gput:Nnx \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
              }
            \tl_put_right:Nx \l__tag_attr_value_tl
              {
                \c_space_tl
                \prop_item:Nn \g__tag_attr_objref_prop {##1}
              }
%      \tl_show:N \l__tag_attr_value_tl
          }
        \tl_put_right:Nx \l__tag_attr_value_tl
          { %[
            \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{]}%
          }
%      \tl_show:N \l__tag_attr_value_tl
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { A }
          { \l__tag_attr_value_tl }
     },
   }
⟨/package⟩
```

(*End definition for* `attribute`. *This function is documented on page 64.*)

Part VIII

The **tagpdf-luatex.def** Driver for luatex Part of the tagpdf package

```
⟨@@=tag⟩
⟨*luatex⟩
\ProvidesExplFile {tagpdf-luatex.def} {2021-07-03} {0.91}
  {tagpdf~driver~for~luatex}
```

1 Loading the lua

The space code requires that the fall back font has been loaded and initialized, so we force that first. But perhaps this could be done in the kernel.

```
{
  \fontencoding{TU}\fontfamily{lmr}\fontseries{m}\fontshape{n}\fontsize{10pt}{10pt}\selectfon
}
\lua_now:e { tagpdf=require('tagpdf.lua') }
```

The following defines wrappers around prop and seq commands to store the data also in lua tables. I probably want also lua tables I put them in the ltx.@@.tables namespaces The tables will be named like the variables but without backslash To access such a table with a dynamical name create a string and then use ltx.@@.tables[string] Old code, I'm not quite sure if this was a good idea. Now I have mix of table in ltx.@@.tables and ltx.@@.mc/struct. And a lot is probably not needed. TODO: this should be cleaned up, but at least roles are currently using the table!

__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_protected:Npn \__tag_prop_new:N #1
  {
    \prop_new:N #1
    \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
  }


\cs_set_protected:Npn \__tag_seq_new:N #1
  {
    \seq_new:N #1
    \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} }
  }


\cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
  {
    \prop_gput:Nnn #1 { #2 } { #3 }
    \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 ["#2"] = "#3" }
  }


```

```
\cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
  {
    \seq_gput_right:Nn #1 { #2 }
    \lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }
  }

%Hm not quite sure about the naming

\cs_set:Npn \__tag_seq_item:cn #1 #2
  {
    \lua_now:e { tex.print(ltx.__tag.tables.#1[#2]) }
  }

\cs_set:Npn \__tag_prop_item:cn #1 #2
  {
    \lua_now:e { tex.print(ltx.__tag.tables.#1["#2"]) }
  }

%for debugging commands that show both the seq/prop and the lua tables
\cs_set_protected:Npn \__tag_seq_show:N #1
  {
    \seq_show:N #1
    \lua_now:e { ltx.__tag.trace.log ("lua~sequence~array~\cs_to_str:N#1",1) }
    \lua_now:e { ltx.__tag.trace.show_seq (ltx.__tag.tables.\cs_to_str:N#1) }
  }

\cs_set_protected:Npn \__tag_prop_show:N #1
  {
    \prop_show:N #1
    \lua_now:e {ltx.__tag.trace.log  ("lua~property~table~\cs_to_str:N#1",1) }
    \lua_now:e {ltx.__tag.trace.show_prop (ltx.__tag.tables.\cs_to_str:N#1) }
 }
```

(*End definition for* `\__tag_prop_new:N` *and others.*)

```
⟨/luatex⟩
```

The module declaration

```
⟨*lua⟩
-- tagpdf.lua
-- Ulrike Fischer

local ProvidesLuaModule = {
    name          = "tagpdf",
    version       = "0.91",       --TAGVERSION
    date          = "2021-07-03", --TAGDATE
    description   = "tagpdf lua code",
    license       = "The LATEX Project Public License 1.3c"
}

if luatexbase and luatexbase.provides_module then
  luatexbase.provides_module (ProvidesLuaModule)
end

--[[
The code has quite probably a number of problems
 - more variables should be local instead of global
 - the naming is not always consistent due to the development of the code
 - the traversing of the shipout box must be tested with more complicated setups
 - it should probably handle more node types
 -
--]]

```

Some comments about the lua structure.

```
--[[
the main table is named ltx.__tag. It contains the functions and also the data
collected during the compilation.

ltx.__tag.mc     will contain mc connected data.
ltx.__tag.struct will contain structure related data.
ltx.__tag.page   will contain page data
ltx.__tag.tables contains also data from mc and struct (from older code). This needs cleaning
             There are certainly dublettes, but I don't dare yet ...
ltx.__tag.func   will contain (public) functions.
ltx.__tag.trace  will contain tracing/loging functions.
local funktions starts with __
functions meant for users will be in ltx.tag

functions
 ltx.__tag.func.get_num_from (tag):    takes a tag (string) and returns the id number
 ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
 ltx.__tag.func.get_tag_from (num):    takes a num and returns the tag
 ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
 ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
 ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
 ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
 ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number o
 ltx.__tag.func.store_struct_mcabs (structnum,mcnum): stores relations structnum<->mcnum (abs)
 ltx.__tag.func.mc_insert_kids (mcnum): inserts the /K entries for mcnum by wandering through
 ltx.__tag.func.mark_page_elements(box,mcpagecnt,mccntprev,mcopen,name,mctypeprev) : the main
 ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EM
 ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this
 ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
 ltx.__tag.func.pdf_object_ref(name): outputs the object reference for the object name
 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of pos
 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log level
 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current lo
 ltx.__tag.trace.show_seq: shows a sequence (array)
 ltx.__tag.trace.show_struct_data (num): shows data of structure num
 ltx.__tag.trace.show_prop: shows a prop
 ltx.__tag.trace.log
 ltx.__tag.trace.showspaces : boolean
--]]

```

This set-ups the main attribute registers. The mc_type attribute stores the type (P, Span etc) encoded as a num, The mc_cnt attribute stores the absolute number and allows so to see if a node belongs to the same mc-chunk.

The interwordspace attr is set by the function `@@_mark_spaces`, and marks the place where spaces should be inserted. The interwordfont attr is set by the function `@@_mark_spaces` too and stores the font, so that we can decide which font to use for the real space char.

```
local mctypeattributeid  = luatexbase.new_attribute ("g__tag_mc_type_attr")
local mccntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
local iwspaceattributeid = luatexbase.new_attribute ("g__tag_interwordspace_attr")
local iwfontattributeid  = luatexbase.new_attribute ("g__tag_interwordfont_attr")
```

with this token we can query the state of the boolean and so detect if unmarked nodes should be marked as attributes

```
local tagunmarkedbool= token.create("g__tag_tagunmarked_bool")
local truebool       = token.create("c_true_bool")
```

Now a number of local versions from global tables. Not all is perhaps needed, most node variants were copied from lua-debug.

```
local catlatex       = luatexbase.registernumber("catcodetable@latex")
local tableinsert    = table.insert
local nodeid           = node.id
local nodecopy         = node.copy
local nodegetattribute = node.get_attribute
local nodesetattribute = node.set_attribute
local nodehasattribute = node.has_attribute
local nodenew          = node.new
local nodetail         = node.tail
local nodeslide        = node.slide
local noderemove       = node.remove
local nodetraverseid   = node.traverse_id
local nodetraverse     = node.traverse
local nodeinsertafter  = node.insert_after
local nodeinsertbefore = node.insert_before
local pdfpageref       = pdf.pageref

local HLIST          = node.id("hlist")
local VLIST          = node.id("vlist")
local RULE           = node.id("rule")
local DISC           = node.id("disc")
local GLUE           = node.id("glue")
local GLYPH          = node.id("glyph")
local KERN           = node.id("kern")
local PENALTY        = node.id("penalty")
local LOCAL_PAR      = node.id("local_par")
local MATH           = node.id("math")
```

Now we setup the main table structure. ltx is used by other latex code too!

```
ltx             = ltx        or { }
ltx.__tag          = ltx.__tag        or { }
ltx.__tag.mc       = ltx.__tag.mc     or  { } -- mc data
ltx.__tag.struct   = ltx.__tag.struct or  { } -- struct data
ltx.__tag.tables   = ltx.__tag.tables or  { } -- tables created with new prop and new seq.
                                        -- wasn't a so great idea ...
                                        -- g__tag_role_tags_seq used by tag<-> is in this tabl
ltx.__tag.page     = ltx.__tag.page   or  { } -- page data, currently only i->{0->mcnum,1->mcn
ltx.__tag.trace    = ltx.__tag.trace  or  { } -- show commands
ltx.__tag.func     = ltx.__tag.func   or  { } -- functions
ltx.__tag.conf     = ltx.__tag.conf   or  { } -- configuration variables
```

2 Logging functions

__tag_log
ltx.__tag.trace.log

This rather simple log function takes as argument a message (string) and a number and will output the message to the log/terminal if the current loglevel is greater or equal than num.

```
local __tag_log =
 function (message,loglevel)
  if (loglevel or 3) <= tex.count["l__tag_loglevel_int"] then
   texio.write_nl("tagpdf: ".. message)
  end
 end

ltx.__tag.trace.log = __tag_log
```

(*End definition for* `__tag_log` *and* `ltx.__tag.trace.log`.)

ltx.__tag.trace.show_seq

This shows the content of a seq as stored in the tables table. It is used by the `\@@_seq_show:N` function. It is not used in user commands, only for debugging, and so requires log level >0.

```
function ltx.__tag.trace.show_seq (seq)
 if (type(seq) == "table") then
  for i,v in ipairs(seq) do
   __tag_log ("[" .. i .. "] => " .. tostring(v),1)
  end
  else
   __tag_log ("sequence " .. tostring(seq) .. " not found",1)
  end
end
```

(*End definition for* `ltx.__tag.trace.show_seq`.)

__tag_pairs_prop
ltx.__tag.trace.show_prop

This shows the content of a prop as stored in the tables table. It is used by the `\@@_prop_show:N` function.

```
local __tag_pairs_prop =
 function  (prop)
      local a = {}
      for n in pairs(prop) do tableinsert(a, n) end
      table.sort(a)
      local i = 0                -- iterator variable
      local iter = function ()   -- iterator function
        i = i + 1
        if a[i] == nil then return nil
        else return a[i], prop[a[i]]
        end
      end
      return iter
  end


function ltx.__tag.trace.show_prop (prop)
 if (type(prop) == "table") then
```

```
   for i,v in __tag_pairs_prop (prop) do
     __tag_log ("[" .. i .. "] => " .. tostring(v),1)
   end
  else
    __tag_log ("prop " .. tostring(prop) .. " not found or not a table",1)
  end
  end
```

(*End definition for* `__tag_pairs_prop` *and* `ltx.__tag.trace.show_prop`.)

`ltx.__tag.trace.show_mc_data` This shows some data for a mc given by `num`. If something is shown depends on the log level. The function is used by the following function and then in `\ShowTagging`

```
function ltx.__tag.trace.show_mc_data (num,loglevel)
 if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
  for k,v in pairs(ltx.__tag.mc[num]) do
   __tag_log  ("mc"..num..": "..tostring(k).."=>"..tostring(v),loglevel)
  end
  if ltx.__tag.mc[num]["kids"] then
  __tag_log ("mc" .. num .. " has " .. #ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
   for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
    __tag_log ("mc ".. num .. " kid "..k.." =>" .. v.kid.." on page " ..v.page,loglevel)
   end
  end
 else
  __tag_log  ("mc"..num.." not found",loglevel)
 end
end
```

(*End definition for* `ltx.__tag.trace.show_mc_data`.)

`ltx.__tag.trace.show_all_mc_data` This shows data for the mc's between `min` and `max` (numbers). It is used by the `\ShowTagging` function.

```
function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
 for i = min, max do
  ltx.__tag.trace.show_mc_data (i,loglevel)
 end
 texio.write_nl("")
end
```

(*End definition for* `ltx.__tag.trace.show_all_mc_data`.)

`ltx.__tag.trace.show_struct_data` This function shows some struct data. Unused but kept for debugging.

```
function ltx.__tag.trace.show_struct_data (num)
 if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
  for k,v in ipairs(ltx.__tag.struct[num]) do
   __tag_log  ("struct "..num..": "..tostring(k).."=>"..tostring(v),1)
  end
 else
  __tag_log   ("struct "..num.." not found ",1)
 end
end
```

(*End definition for* `ltx.__tag.trace.show_struct_data`.)

3 Helper functions

3.1 Retrieve data functions

__tag_get_mc_cnt_type_tag

This takes a node as argument and returns the mc-cnt, the mc-type and and the tag (calculated from the mc-cnt.

```
local __tag_get_mc_cnt_type_tag = function (n)
  local mccnt      =  nodegetattribute(n,mccntattributeid)  or -1
  local mctype     =  nodegetattribute(n,mctypeattributeid)  or -1
  local tag        =  ltx.__tag.func.get_tag_from(mctype)
  return mccnt,mctype,tag
end
```

(*End definition for* `__tag_get_mc_cnt_type_tag`.)

__tag_get_mathsubtype

This function allows to detect if we are at the begin or the end of math. It takes as argument a mathnode.

```
local function __tag_get_mathsubtype  (mathnode)
 if mathnode.subtype == 0 then
  subtype = "beginmath"
 else
  subtype = "endmath"
 end
 return subtype
end
```

(*End definition for* `__tag_get_mathsubtype`.)

__tag_get_num_from
ltx.__tag.func.get_num_from
ltx.__tag.func.output_num_from

These functions take as argument a string `tag`, and return the number under which is it recorded (and so the attribute value). The first function outputs the number for lua, while the `output` function outputs to tex.

```
local __tag_get_num_from =
 function (tag)
  if ltx.__tag.tables["g__tag_role_tags_prop"][tag] then
    a= ltx.__tag.tables["g__tag_role_tags_prop"][tag]
  else
    a= -1
  end
  return a
 end

ltx.__tag.func.get_num_from = __tag_get_num_from

function ltx.__tag.func.output_num_from (tag)
  local num = __tag_get_num_from (tag)
  tex.sprint(catlatex,num)
  if num == -1 then
   __tag_log ("Unknown tag "..tag.." used")
  end
end
```

(*End definition for* `__tag_get_num_from`, `ltx.__tag.func.get_num_from`, *and* `ltx.__tag.func.output_num_from`.)

__tag_get_tag_from
ltx.__tag.func.get_tag_from
ltx.__tag.func.output_tag_from

These functions are the opposites to the previous function: they take as argument a number (the attribute value) and return the string `tag`. The first function outputs the number for lua, while the `output` function outputs to tex.

```
local __tag_get_tag_from =
 function  (num)
  if ltx.__tag.tables["g__tag_role_tags_seq"][num] then
   a = ltx.__tag.tables["g__tag_role_tags_seq"][num]
  else
   a= "UNKNOWN"
  end
 return a
end

ltx.__tag.func.get_tag_from = __tag_get_tag_from

function ltx.__tag.func.output_tag_from (num)
  tex.sprint(catlatex,__tag_get_tag_from (num))
end
```

(*End definition for* `__tag_get_tag_from` *,* `ltx.__tag.func.get_tag_from` *, and* `ltx.__tag.func.output_tag_from`*.*)

ltx.__tag.func.store_mc_data

This function stores for `key`=`data` for mc-chunk `num`. It is used in the tagpdf-mc code, to store for example the tag string, and the raw options.

```
function ltx.__tag.func.store_mc_data (num,key,data)
 ltx.__tag.mc[num] = ltx.__tag.mc[num] or { }
 ltx.__tag.mc[num][key] = data
 __tag_log  ("INFO TEX-STORE-MC-DATA: "..num.." => "..tostring(key).." => "..tostring(data),3
end
```

(*End definition for* `ltx.__tag.func.store_mc_data`*.*)

ltx.__tag.func.store_mc_label

This function stores the `label`=`num` relationship in the `labels` subtable. TODO: this is probably unused and can go.

```
function ltx.__tag.func.store_mc_label (label,num)
 ltx.__tag.mc["labels"] = ltx.__tag.mc["labels"] or { }
 ltx.__tag.mc.labels[label] = num
end
```

(*End definition for* `ltx.__tag.func.store_mc_label`*.*)

ltx.__tag.func.store_mc_kid

This function is used in the traversing code. It stores a sub-chunk of a mc `mcnum` into the `kids` table.

```
function ltx.__tag.func.store_mc_kid (mcnum,kid,page)
 ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.." => " .. kid.." on page " .. page,3)
 ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
 local kidtable = {kid=kid,page=page}
 tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
end
```

(*End definition for* `ltx.__tag.func.store_mc_kid`*.*)

ltx.__tag.func.mc_num_of_kids This function returns the number of kids a mc `mcnum` has. We need to account for the case that a mc can have no kids.

```
function ltx.__tag.func.mc_num_of_kids (mcnum)
 local num = 0
 if ltx.__tag.mc[mcnum] and ltx.__tag.mc[mcnum]["kids"] then
   num = #ltx.__tag.mc[mcnum]["kids"]
 end
 ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
 return num
end
```

(*End definition for* `ltx.__tag.func.mc_num_of_kids`.)

3.2 Functions to insert the pdf literals

`__tag_insert_emc_node` This insert the emc node.

```
local function __tag_insert_emc_node (head,current)
 local emcnode = nodenew("whatsit","pdf_literal")
       emcnode.data = "EMC"
       emcnode.mode=1
       head = node.insert_before(head,current,emcnode)
 return head
end
```

(*End definition for* `__tag_insert_emc_node`.)

`__tag_insert_bmc_node` This inserts a simple bmc node

```
local function __tag_insert_bmc_node (head,current,tag)
 local bmcnode = nodenew("whatsit","pdf_literal")
       bmcnode.data = "/"..tag.." BMC"
       bmcnode.mode=1
       head = node.insert_before(head,current,bmcnode)
 return head
end
```

(*End definition for* `__tag_insert_bmc_node`.)

`__tag_insert_bdc_node` This inserts a bcd node with a fix dict. TODO: check if this is still used, now that we create properties.

```
local function __tag_insert_bdc_node (head,current,tag,dict)
 local bdcnode = nodenew("whatsit","pdf_literal")
       bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
       bdcnode.mode=1
       head = node.insert_before(head,current,bdcnode)
 return head
end
```

(*End definition for* `__tag_insert_bdc_node`.)

`__tag_pdf_object_ref`
ltx.__tag.func.pdf_object_ref This allows to reference a pdf object reserved with the l3pdf command by name. The return value is `n 0 R`, if the object doesn't exist, n is 0. TODO: is uses internal l3pdf commands, this should be properly supported by l3pdf

```
local function __tag_pdf_object_ref (name)
   local tokenname = 'c__pdf_backend_object_'..name..'_int'
   local object = token.create(tokenname).index..' 0 R'
   return object
end
ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref
```

(*End definition for* `__tag_pdf_object_ref` *and* `ltx.__tag.func.pdf_object_ref`.)

4 Function for the real space chars

`__tag_show_spacemark`

A debugging function, it is used to inserts red color markers in the places where space chars can go, it can have side effects so not always reliable, but ok.

```
local function __tag_show_spacemark (head,current,color,height)
 local markcolor = color or "1 0 0"
 local markheight = height or 10
 local pdfstring = node.new("whatsit","pdf_literal")
       pdfstring.data =
       string.format("q "..markcolor.." RG "..markcolor.." rg 0.4 w 0 %g m 0 %g l S Q",-
  3,markheight)
       head = node.insert_after(head,current,pdfstring)
 return head
end
```

(*End definition for* `__tag_show_spacemark`.)

`__tag_fakespace`
`ltx.__tag.func.fakespace`

This is used to define a lua version of `\pdffakespace`

```
local function __tag_fakespace()
   tex.setattribute(iwspaceattributeid,1)
   tex.setattribute(iwfontattributeid,font.current())
end
ltx.__tag.func.fakespace = __tag_fakespace
```

(*End definition for* `__tag_fakespace` *and* `ltx.__tag.func.fakespace`.)

`__tag_mark_spaces`

a function to mark up places where real space chars should be inserted. It only sets attributes, these are then be used in a later traversing which inserts the actual spaces. When space handling is activated this function is inserted in some callbacks.

```
--[[ a function to mark up places where real space chars should be inserted
     it only sets an attribute.
--]]

local function __tag_mark_spaces (head)
  local inside_math = false
  for n in nodetraverse(head) do
    local id = n.id
    if id == GLYPH then
      local glyph = n
      if glyph.next and (glyph.next.id == GLUE)
        and not inside_math  and (glyph.next.width >0)
      then
        nodesetattribute(glyph.next,iwspaceattributeid,1)
        nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
      -- for debugging
       if ltx.__tag.trace.showspaces then
```

```
        __tag_show_spacemark (head,glyph)
       end
      elseif glyph.next and (glyph.next.id==KERN) and not inside_math then
       local kern = glyph.next
       if kern.next and (kern.next.id== GLUE)  and (kern.next.width >0)
       then
        nodesetattribute(kern.next,iwspaceattributeid,1)
        nodesetattribute(kern.next,iwfontattributeid,glyph.font)
       end
      end
     --  look also back
     if glyph.prev and (glyph.prev.id == GLUE)
        and not inside_math
        and (glyph.prev.width >0)
        and not nodehasattribute(glyph.prev,iwspaceattributeid)
      then
        nodesetattribute(glyph.prev,iwspaceattributeid,1)
        nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
      -- for debugging
       if ltx.__tag.trace.showspaces then
        __tag_show_spacemark (head,glyph)
       end
      end
    elseif id == PENALTY then
      local glyph = n
      -- ltx.__tag.trace.log ("PENALTY ".. n.subtype.."VALUE"..n.penalty,3)
      if glyph.next and (glyph.next.id == GLUE)
        and not inside_math  and (glyph.next.width >0) and n.subtype==0
      then
        nodesetattribute(glyph.next,iwspaceattributeid,1)
      --  nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
      -- for debugging
       if ltx.__tag.trace.showspaces then
        __tag_show_spacemark (head,glyph)
       end
      end
    elseif id == MATH then
      inside_math = (n.subtype == 0)
    end
  end
  return head
end
```

(*End definition for* `__tag_mark_spaces`.)

`__tag_activate_mark_space`
`ltx.__tag.func.markspaceon`
`ltx.__tag.func.markspaceoff`

Theses functions add/remove the function which marks the spaces to the callbacks `pre_linebreak_filter` and `hpack_filter`

```
local function __tag_activate_mark_space ()
 if not luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
  luatexbase.add_to_callback("pre_linebreak_filter",__tag_mark_spaces,"markspaces")
  luatexbase.add_to_callback("hpack_filter",__tag_mark_spaces,"markspaces")
 end
end

```

```
ltx.__tag.func.markspaceon=__tag_activate_mark_space

local function __tag_deactivate_mark_space ()
 if luatexbase.in_callback ("pre_linebreak_filter","markspaces") then
 luatexbase.remove_from_callback("pre_linebreak_filter","markspaces")
 luatexbase.remove_from_callback("hpack_filter","markspaces")
 end
end

ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
```

(*End definition for* `__tag_activate_mark_space` , `ltx.__tag.func.markspaceon` , *and* `ltx.__tag.func.markspaceoff`.)

`default_space_char`
`default_fontid`

We need two local variable to setup a default space char.

```
local default_space_char = node.new(GLYPH)
local default_fontid     = font.id("TU/lmr/m/n/10")
default_space_char.char  = 32
default_space_char.font  = default_fontid
```

(*End definition for* `default_space_char` *and* `default_fontid`. *These functions are documented on page* **??**.)

`__tag_space_chars_shipout`
`ltx.__tag.func.space_chars_shipout`

These is the main function to insert real space chars. It inserts a glyph before every glue which has been marked previously. The attributes are copied from the glue, so if the tagging is done later, it will be tagged like it.

```
local function __tag_space_chars_shipout (box)
 local head = box.head
  if head then
    for n in node.traverse(head) do
      local spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
      if n.id == HLIST  then -- enter the hlist
         __tag_space_chars_shipout (n)
      elseif n.id == VLIST then -- enter the vlist
         __tag_space_chars_shipout (n)
      elseif n.id == GLUE then
        if ltx.__tag.trace.showspaces and spaceattr==1  then
          __tag_show_spacemark (head,n,"0 1 0")
        end
        if spaceattr==1  then
          local space
          local space_char = node.copy(default_space_char)
          local curfont    = nodegetattribute(n,iwfontattributeid)
          ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
          if curfont and luaotfload.aux.slot_of_name(curfont,"space") then
            space_char.font=curfont
          end
          head, space = node.insert_before(head, n, space_char) --
          n.width     = n.width - space.width
          space.attr  = n.attr
        end
      end
    end
  end
end
```

```

function ltx.__tag.func.space_chars_shipout (box)
  __tag_space_chars_shipout (box)
end
```

(*End definition for* `__tag_space_chars_shipout` *and* `ltx.__tag.func.space_chars_shipout`.)

5 Function for the tagging

ltx.__tag.func.mc_insert_kids

This is the main function to insert the K entry into a StructElem object. It is used in tagpdf-mc-luacode module. The `single` attribute allows to handle the case that a single mc on the tex side can have more than one kid after the processing here, and so we get the correct array/non array setup.

```
function ltx.__tag.func.mc_insert_kids (mcnum,single)
  if ltx.__tag.mc[mcnum] then
  ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID-TEST: " .. mcnum,4)
   if ltx.__tag.mc[mcnum]["kids"] then
    if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
     tex.sprint("[")
    end
    for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
     local kidnum  = kidstable["kid"]
     local kidpage = kidstable["page"]
     local kidpageobjnum = pdfpageref(kidpage)
     ltx.__tag.trace.log("INFO TEX-MC-INSERT-KID: " .. mcnum ..
                         " insert KID " ..i..
                         " with num " .. kidnum ..
                         " on page " .. kidpage.."/"..kidpageobjnum,3)
     tex.sprint(catlatex,"<</Type /MCR /Pg "..kidpageobjnum .. " 0 R /MCID "..kidnum.. ">> "
    end
    if #ltx.__tag.mc[mcnum]["kids"] > 1 and single==1 then
     tex.sprint("]")
    end
   else
    -- this is typically not a problem, e.g. empty hbox in footer/header can
    -- trigger this warning.
    ltx.__tag.trace.log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
    if single==1 then
      tex.sprint("null")
    end
   end
  else
   ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
  end
end
```

(*End definition for* `ltx.__tag.func.mc_insert_kids`.)

ltx.__tag.func.store_struct_mcabs

This function is used in the tagpdf-mc-luacode. It store the absolute count of the mc into the current structure. This must be done ordered.

```
function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
 ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
 ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
 -- a structure can contain more than on mc chunk, the content should be ordered
 tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
 ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: "..
                  mcnum.." inserted in struct "..structnum,3)
 -- but every mc can only be in one structure
 ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
 ltx.__tag.mc[mcnum]["parent"] = structnum
end

```

(*End definition for* `ltx.__tag.func.store_struct_mcabs`.)

ltx.__tag.func.store_mc_in_page

This is used in the traversing code and stores the relation between abs count and page count.

```
-- pay attention: lua counts arrays from 1, tex pages from one
-- mcid and arrays in pdf count from 0.
function ltx.__tag.func.store_mc_in_page (mcnum,mcpagecnt,page)
 ltx.__tag.page[page] = ltx.__tag.page[page] or {}
 ltx.__tag.page[page][mcpagecnt] = mcnum
 ltx.__tag.trace.log("INFO TAG-MC-INTO-PAGE: page " .. page ..
                  ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
end
```

(*End definition for* `ltx.__tag.func.store_mc_in_page`.)

ltx.__tag.func.mark_page_elements

This is the main traversing function. See the lua comment for more details.

```
--[[
    Now follows the core function
    It wades through the shipout box and checks the attributes
    ARGUMENTS
    box: is a box,
    mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed fo
    mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a 
    mcopen: num, records if some bdc/emc is open
    These arguments are only needed for log messages, if not present are replaces by fix stri
    name: string to describe the box
    mctypeprev: num, the type attribute of the previous node/whatever

    there are lots of logging messages currently. Should be cleaned up in due course.
    One should also find ways to make the function shorter.
--]]

function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mccntprev,mcopen,name,mctypeprev)
  local name = name or ("SOMEBOX")
  local mctypeprev = mctypeprev or -1
  local abspage = status.total_pages + 1  -- the real counter is increased
                                          -- inside the box so one off
                                          -- if the callback is not used. (???)
  ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
  ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
                     " prev "..mccntprev ..
                     " type prev "..mctypeprev,4)
  ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ".. tostring(name)..
                     " TYPE ".. node.type(node.getid(box)),3)
```

```
local head = box.head -- ShipoutBox is a vlist?
if head then
  mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
  ltx.__tag.trace.log ("INFO TAG-HEAD: " ..
                     node.type(node.getid(head))..
                     " MC"..tostring(mccnthead)..
                     " => TAG " .. tostring(mctypehead)..
                     " => ".. tostring(taghead),3)
else
  ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is "..
                      tostring(head),3)
end
for n in node.traverse(head) do
  local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
  local spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
  ltx.__tag.trace.log ("INFO TAG-NODE: "..
                      node.type(node.getid(n))..
                     " MC".. tostring(mccnt)..
                     " => TAG ".. tostring(mctype)..
                     " => " ..  tostring(tag),3)
  if n.id == HLIST
  then -- enter the hlist
   mcopen,mcpagecnt,mccntprev,mctypeprev=
    ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL HLIST",mctypep
  elseif n.id == VLIST then -- enter the vlist
   mcopen,mcpagecnt,mccntprev,mctypeprev=
    ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctypep
  elseif n.id == GLUE then       -- at glue real space chars are inserted, but this has
                                 -- been done if the previous shipout wandering, so here it
  elseif n.id == LOCAL_PAR then  -- local_par is ignored
  elseif n.id == PENALTY then    -- penalty is ignored
  elseif n.id == KERN then       -- kern is ignored
   ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
     node.type(node.getid(n)).." "..n.subtype,4)
  else
   -- math is currently only logged.
   -- we could mark the whole as math
   -- for inner processing the mlist_to_hlist callback is probably needed.
   if n.id == MATH then
    ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: "..
      node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
   end
   -- endmath
   ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..
            mccnt.." prev "..mccntprev,4)
   if mccnt~=mccntprev then -- a new mc chunk
    ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: "..
                         node.type(node.getid(n))..
                        " MC"..tostring(mccnt)..
                        " <=> PREVIOUS "..tostring(mccntprev),4)
    if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
     box.list=__tag_insert_emc_node (box.list,n)
     mcopen = mcopen - 1
     ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: " ..
       mcpagecnt .. " MCOPEN = " .. mcopen,3)
     if mcopen ~=0 then
      ltx.__tag.trace.log ("WARN TAG-OPEN-MC: " .. mcopen,1)
     end
    end
    if ltx.__tag.mc[mccnt] then
     if ltx.__tag.mc[mccnt]["artifact"] then
      ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: "..
                        tostring(ltx.__tag.mc[mccnt]["artifact"]),3)
      if ltx.__tag.mc[mccnt]["artifact"] == "" then
       box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
      else
       box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mcc
      end
     else
      ltx.__tag.trace.log("INFO TAG-INSERT-TAG: "..
                        tostring(tag),3)
      mcpagecnt = mcpagecnt +1
      ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
      local dict= "/MCID "..mcpagecnt
      if ltx.__tag.mc[mccnt]["raw"] then
       ltx.__tag.trace.log("INFO TAG-USE-RAW: "..
         tostring(ltx.__tag.mc[mccnt]["raw"]),3)
       dict= dict .. " " .. ltx.__tag.mc[mccnt]["raw"]
      end
      if ltx.__tag.mc[mccnt]["alt"] then
       ltx.__tag.trace.log("INFO TAG-USE-ALT: "..
          tostring(ltx.__tag.mc[mccnt]["alt"]),3)
       dict= dict .. " " .. ltx.__tag.mc[mccnt]["alt"]
      end
      if ltx.__tag.mc[mccnt]["actualtext"] then
       ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: "..
         tostring(ltx.__tag.mc[mccnt]["actualtext"]),3)
       dict= dict .. " " .. ltx.__tag.mc[mccnt]["actualtext"]
      end
      box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
      ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
      ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
      ltx.__tag.trace.show_mc_data (mccnt,3)
     end
     mcopen = mcopen + 1
    else
     if tagunmarkedbool.mode == truebool.mode then
      ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2
      box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
      mcopen = mcopen + 1
     else
      ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
     end
    end
    mccntprev = mccnt
   end
  end -- end if
end -- end for
```

```
  if head then
    mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
    ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
                         node.type(node.getid(head))..
                        " MC"..tostring(mccnthead)..
                        " => TAG "..tostring(mctypehead)..
                        " => "..tostring(taghead),4)
  else
    ltx.__tag.trace.log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
  end
  ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX "..
                       tostring(name)..
                      " TYPE ".. node.type(node.getid(box)),4)
 return mcopen,mcpagecnt,mccntprev,mctypeprev
end

```

(*End definition for* `ltx.__tag.func.mark_page_elements`.)

`ltx.__tag.func.mark_shipout`

This is the function used in the callback. Beside calling the traversing function it also checks if there is an open MC-chunk from a page break and insert the needed EMC literal.

```
function ltx.__tag.func.mark_shipout (box)
 mcopen = ltx.__tag.func.mark_page_elements (box,-1,-100,0,"Shipout",-1)
 if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ...
  local emcnode = nodenew("whatsit","pdf_literal")
  local list = box.list
  emcnode.data = "EMC"
  emcnode.mode=1
  if list then
     list = node.insert_after (list,node.tail(list),emcnode)
     mcopen = mcopen - 1
     ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
  else
     ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
  end
  if mcopen ~=0 then
     ltx.__tag.trace.log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
  end
 end
end
```

(*End definition for* `ltx.__tag.func.mark_shipout`.)

6 Parenttree

`ltx.__tag.func.fill_parent_tree_line`
`ltx.__tag.func.output_parenttree`

These functions create the parent tree. The second, main function is used in the tagpdf-tree code. TODO check if the tree code can move into the backend code.

```
function ltx.__tag.func.fill_parent_tree_line (page)
     -- we need to get page-> i=kid -> mcnum -> structnum
     -- pay attention: the kid numbers and the page number in the parent tree start with 0!
    local numsentry =""
    local pdfpage = page-1
    if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
     mcchunks=#ltx.__tag.page[page]
     ltx.__tag.trace.log("INFO PARENTTREE-NUM:  page "..
                   page.." has "..mcchunks.."+1 Elements ",4)
     for i=0,mcchunks do
     -- what does this log??
      ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS:  "..
        ltx.__tag.page[page][i],4)
     end
     if mcchunks == 0 then
      -- only one chunk so no need for an array
      local mcnum  = ltx.__tag.page[page][0]
      local structnum = ltx.__tag.mc[mcnum]["parent"]
      local propname  = "g__tag_struct_"..structnum.."_prop"
      --local objref   =  ltx.__tag.tables[propname]["objref"] or "XXXX"
      local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
      ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF:  =====>"..
        tostring(objref),5)
      numsentry = pdfpage .. " [".. objref .. "]"
      ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
        page.. " num entry = ".. numsentry,3)
     else
      numsentry = pdfpage .. " ["
       for i=0,mcchunks do
        local mcnum  = ltx.__tag.page[page][i]
        local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
        local propname  = "g__tag_struct_"..structnum.."_prop"
        --local objref   =  ltx.__tag.tables[propname]["objref"] or "XXXX"
        local objref = __tag_pdf_object_ref('__tag/struct/'..structnum)
        numsentry = numsentry .. " ".. objref
       end
      numsentry = numsentry .. "] "
      ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page " ..
        page.. " num entry = ".. numsentry,3)
     end
    else
      ltx.__tag.trace.log ("INFO PARENTTREE-NO-DATA: page "..page,3)
    end
    return numsentry
end

function ltx.__tag.func.output_parenttree (abspage)
 for i=1,abspage do
  line = ltx.__tag.func.fill_parent_tree_line (i) .. "^^J"
  tex.sprint(catlatex,line)
 end
end
```

(*End definition for* `ltx.__tag.func.fill_parent_tree_line` *and* `ltx.__tag.func.output_parenttree`.)

```
⟨/lua⟩
```

Part IX
The **tagpdf-roles** module Tags, roles and namesspace code Part of the tagpdf package

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-roles-code} {2021-07-03} {0.91}
 {part of tagpdf - code related to roles and structure names}
⟨/header⟩
```

1 Code related to roles and structure names

1.1 Variables

Tags have both a name (a string) and a number (for the lua attribute). Testing a name is easier with a prop, while accessing with a number is better done with a seq. So both are used and must be kept in sync if a new tag is added. The number is only relevant for the MC type, tags with the same name from different names spaces can have the same number.

\g__tag_role_tags_seq
\g__tag_role_tags_prop

```
⟨*package⟩
\__tag_seq_new:N  \g__tag_role_tags_seq  %to get names (type/NS) from numbers
\__tag_prop_new:N \g__tag_role_tags_prop %to get numbers  from names (type/NS)
```

(*End definition for* \g__tag_role_tags_seq *and* \g__tag_role_tags_prop.)

\g__tag_role_tags_NS_prop

in pdf 2.0 tags belong to a name space. For every tag we store a default name space. The keys are the tags, the value shorthands like pdf2, or mathml. There is no need to access this from lua, so we use the standard prop commands.

```
\prop_new:N    \g__tag_role_tags_NS_prop %to namespace info
```

(*End definition for* \g__tag_role_tags_NS_prop.)

\g__tag_role_NS_prop

The standard names spaces are the following. The keys are the name tagpdf will use, the urls are the identifier in the namespace object.

mathml http://www.w3.org/1998/Math/MathML

pdf2 http://iso.org/pdf2/ssn

pdf http://iso.org/pdf/ssn (default)

user \c__tag_role_userNS_id_str (random id, for user tags)

More namespaces are possible and their objects references and the ones of the namespaces must be collected so that an array can be written to the StructTreeRoot at the end (see tagpdf-tree). We use a prop to store also the object reference as it will be needed rather often.

```
\prop_new:N \g__tag_role_NS_prop % collect namespaces
```

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

We need also a bunch of temporary variables:

\l__tag_role_tag_tmpa_tl
\l__tag_role_tag_namespace_tmpa_tl
\l__tag_role_role_tmpa_tl
\l__tag_role_role_namespace_tmpa_tl

```
\tl_new:N \l__tag_role_tag_tmpa_tl
\tl_new:N \l__tag_role_tag_namespace_tmpa_tl
\tl_new:N \l__tag_role_role_tmpa_tl
\tl_new:N \l__tag_role_role_namespace_tmpa_tl
```

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

1.2 Namesspaces

The following commands setups a names space. Namespace dictionaries can contain an optional `/Schema` and `/RoleMapNS` entry. We only reserve the objects but delay the writing to the finish code, where we can test if the keys and the name spaces are actually needed This commands setups objects for the name space and its rolemap. It also initialize a prop to collect the rolemaps if needed.

`\__tag_role_NS_new:nnn` `\__tag_role_NS_new:nnn{⟨shorthand⟩}{⟨URI-ID⟩}Schema`

__tag_role_NS_new:nnn

```
\cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
  {
    \pdf_object_new:nn {tag/NS/#1}{dict}
    \pdfdict_new:n     {g__tag_role/Namespace_#1_dict}
    \pdf_object_new:nn {__tag/RoleMapNS/#1}{dict}
    \pdfdict_new:n     {g__tag_role/RoleMapNS_#1_dict}
    \pdfdict_gput:nnn
      {g__tag_role/Namespace_#1_dict}
      {Type}
      {/Namespace}
    \pdf_string_from_unicode:nnN{utf8/string}{#2}\l_tmpa_str
    \tl_if_empty:NF \l_tmpa_str
      {
        \pdfdict_gput:nnx
          {g__tag_role/Namespace_#1_dict}
          {NS}
          {\l_tmpa_str}
      }
    %RoleMapNS is added in tree
    \tl_if_empty:nF  {#3}
     {
       \pdfdict_gput:nnx{g__tag_role/Namespace_#1_dict}
        {Schema}{#3}
     }
    \prop_gput:Nnx \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
  }
```

(*End definition for* `\__tag_role_NS_new:nnn`.)

We need an id for the user space. For the tests it should be possible to set it to a fix value. So we use random numbers which can be fixed by setting a seed. We fake a sort of GUID but not try to be really exact as it doesn't matter ...

\c__tag_role_userNS_id_str

```
\str_const:Nx \c__tag_role_userNS_id_str
  { data:,
    \int_to_Hex:n{\int_rand:n {65535}}
    \int_to_Hex:n{\int_rand:n {65535}}
    -
    \int_to_Hex:n{\int_rand:n {65535}}
    -
    \int_to_Hex:n{\int_rand:n {65535}}
    -
    \int_to_Hex:n{\int_rand:n {65535}}
    -
    \int_to_Hex:n{\int_rand:n {16777215}}
    \int_to_Hex:n{\int_rand:n {16777215}}
  }
```

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

Now we setup the standard names spaces. Currently only if we detect pdf2.0 but this will perhaps have to change if the structure code gets to messy.

```
\pdf_version_compare:NnT > {1.9}
  {
    \__tag_role_NS_new:nnn {pdf}   {http://iso.org/pdf/ssn}{}
    \__tag_role_NS_new:nnn {pdf2}  {http://iso.org/pdf2/ssn}{}
    \__tag_role_NS_new:nnn {mathml}{http://www.w3.org/1998/Math/MathML}{}
    \exp_args:Nnx
    \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{}
  }
```

1.3 Data

In this section we setup the standard data. At first the list of structure types. We split them in three lists, the tags with which are both in the pdf and pdf2 namespace, the one only in pdf and the one with the tags only in pdf2. We also define a rolemap for the pdfII only type to pdf so that they can always be used.

\c__tag_role_sttags_pdf_pdfII_clist
\c__tag_role_sttags_only_pdf_clist
\c__tag_role_sttags_only_pdfII_clist
\c__tag_role_sttags_mathml_clist
\c__tag_role_sttags_pdfII_to_pdf_prop

```
%
\clist_const:Nn \c__tag_role_sttags_pdf_pdfII_clist
  {
    Document,   %A complete document. This is the root element
                %of any structure tree containing
                %multiple parts or multiple articles.
    Part,       %A large-scale division of a document.
    Sect,       %A container for grouping related content elements.
    Div,        %A generic block-level element or group of elements
    Caption,    %A brief portion of text describing a table or figure.
    Index,
    NonStruct,  %probably not needed
    H,
    H1,
    H2,
    H3,
    H4,
    H5,
    H6,
    P,
    L,            %list
    LI,           %list item (around label and list item body)
    Lbl,          %list label
    LBody,        %list item body
    Table,
    TR,           %table row
    TH,           %table header cell
    TD,           %table data cell
    THead,        %table header (n rows)
    TBody,        %table rows
    TFoot,        %table footer
    Span,         %generic inline marker
    Link,         %
    Annot,
    Figure,
    Formula,
    Form,
    % ruby warichu etc ..
    Ruby,
    RB,
    RT,
    Warichu,
    WT,
    WP,
    Artifact % only MC-tag ?...
  }

\clist_const:Nn \c__tag_role_sttags_only_pdf_clist
 {
   Art,        %A relatively self-contained body of text
               %constituting a single narrative or exposition
   BlockQuote, %A portion of text consisting of one or more paragraphs
               %attributed to someone other than the author of the
               %surrounding text.
   TOC,        %A list made up of table of contents item entries
               %(structure tag TOCI; see below) and/or other
               %nested table of contents entries
   TOCI,       %An individual member of a table of contents.
               %This entry's children can be any of the following structure  tags:
               %Lbl,Reference,NonStruct,P,TOC
   Index,
   Private,
   Quote,       %inline quote
   Note,        %footnote, endnote. Lbl can be child
   Reference,   %A citation to content elsewhere in the document.
   BibEntry,    %bibentry
   Code
 }

\clist_const:Nn \c__tag_role_sttags_only_pdfII_clist
 {
   DocumentFragment
   ,Aside
   ,H7
   ,H8
   ,H9
   ,H10
   ,Title
   ,FENote
   ,Sub
   ,Em
   ,Strong
   ,Artifact
 }

\clist_const:Nn \c__tag_role_sttags_mathml_clist
 {
   abs
   ,and
   ,annotation
   ,apply
   ,approx
   ,arccos
   ,arccosh
   ,arccot
   ,arccoth
   ,arccsc
   ,arccsch
   ,arcsec
   ,arcsech
   ,arcsin
   ,arcsinh
   ,arctan
   ,arctanh
   ,arg
   ,bind
   ,bvar
   ,card
   ,cartesianproduct
   ,cbytes
   ,ceiling
   ,cerror
   ,ci
   ,cn
   ,codomain
   ,complexes
   ,compose
   ,condition
   ,conjugate
   ,cos
   ,cosh
   ,cot
   ,coth
   ,cs
   ,csc
```

```
,csch
,csymbol
,curl
,declare
,degree
,determinant
,diff
,divergence
,divide
,domain
,domainofapplication
,emptyset
,eq
,equivalent
,eulergamma
,exists
,exp
,exponentiale
,factorial
,factorof
,false
,floor
,fn
,forall
,gcd
,geq
,grad
,gt
,ident
,image
,imaginary
,imaginaryi
,implies
,in
,infinity
,int
,integers
,intersect
,interval
,inverse
,lambda
,laplacian
,lcm
,leq
,limit
,ln
,log
,logbase
,lowlimit
,lt
,maction
,maligngroup
,malignmark
,math
```

```
,matrix
,matrixrow
,max
,mean
,median
,menclose
,merror
,mfenced
,mfrac
,mglyph
,mi
,min
,minus
,mlabeledtr
,mlongdiv
,mmultiscripts
,mn
,mo
,mode
,moment
,momentabout
,mover
,mpadded
,mphantom
,mprescripts
,mroot
,mrow
,ms
,mscarries
,mscarry
,msgroup
,msline
,mspace
,msqrt
,msrow
,mstack
,mstyle
,msub
,msubsup
,msup
,mtable
,mtd
,mtext
,mtr
,munder
,munderover
,naturalnumbers
,neq
,none
,not
,notanumber
,notin
,notprsubset
,notsubset
```

```
    ,or
    ,otherwise
    ,outerproduct
    ,partialdiff
    ,pi
    ,piece
    ,piecewise
    ,plus
    ,power
    ,primes
    ,product
    ,prsubset
    ,quotient
    ,rationals
    ,real
    ,reals
    ,reln
    ,rem
    ,root
    ,scalarproduct
    ,sdev
    ,sec
    ,sech
    ,selector
    ,semantics
    ,sep
    ,set
    ,setdiff
    ,share
    ,sin
    ,sinh
    ,subset
    ,sum
    ,tan
    ,tanh
    ,tendsto
    ,times
    ,transpose
    ,true
    ,union
    ,uplimit
    ,variance
    ,vector
    ,vectorproduct
    ,xor
 }

\prop_const_from_keyval:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
  {
    DocumentFragment = Art,
    Aside = Note,
    Title = H1,
    Sub   = Span,
    H7    = H6 ,
    H8    = H6 ,
    H9    = H6 ,
    H10   = H6,
    FENote= Note,
    Em    = Span,
    Strong= Span,
  }
```

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

We fill the structure tags in to the seq. We allow all pdf1.7 and pdf2.0, and role map if needed the 2.0 tags.

```
% get tag name from number: \seq_item:Nn \g__tag_role_tags_seq { n }
% get tag number from name: \prop_item:Nn \g__tag_role_tags_prop { name }

\clist_map_inline:Nn \c__tag_role_sttags_pdf_pdfII_clist
  {
    \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
    \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ pdf2 }
  }
\clist_map_inline:Nn \c__tag_role_sttags_only_pdf_clist
  {
    \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
    \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ pdf }
  }
\clist_map_inline:Nn \c__tag_role_sttags_only_pdfII_clist
  {
    \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
    \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ pdf2 }
  }
\pdf_version_compare:NnT > {1.9}
  {
     \clist_map_inline:Nn \c__tag_role_sttags_mathml_clist
       {
         \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
         \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ mathml }
       }
  }
```

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

```
\int_step_inline:nnnn { 1 }{ 1 }{ \seq_count:N \g__tag_role_tags_seq }
  {
    \__tag_prop_gput:Nxn \g__tag_role_tags_prop
      {
        \seq_item:Nn \g__tag_role_tags_seq  { #1 }
      }
      { #1 }
  }
```

1.4 Adding new tags and rolemapping

1.4.1 pdf 1.7 and earlier

With this versions only RoleMap is filled. At first the dictionary:

g__tag_role/RoleMap_dict

```
\pdfdict_new:n {g__tag_role/RoleMap_dict}
```

(*End definition for* `g__tag_role/RoleMap_dict`.)

__tag_role_add_tag:nn

The pdf 1.7 version has only two arguments: new and rolemap name. To make pdf 2.0 types usable we directly define a rolemapping for them.

```
\cs_new_protected:Nn \__tag_role_add_tag:nn %(new) name, reference to old
  {
    \prop_if_in:NnF \g__tag_role_tags_prop {#1}
      {
        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
          {
            \msg_info:nnn { tag }{new-tag}{#1}
          }
         \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
         \__tag_prop_gput:Nnx \g__tag_role_tags_prop    { #1 }
           {
             \seq_count:N \g__tag_role_tags_seq
           }
         \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ user }
      }
    \__tag_check_add_tag_role:nn {#1}{#2}
    \tl_if_empty:nF { #2 }
      {
        \pdfdict_gput:nnx {g__tag_role/RoleMap_dict}
          {#1}
          {\pdf_name_from_unicode_e:n{#2}}
      }
  }
\cs_generate_variant:Nn \__tag_role_add_tag:nn {VV}

\pdf_version_compare:NnT < {2.0}
  {
     \prop_map_inline:Nn \c__tag_role_sttags_pdfII_to_pdf_prop
       {
         \__tag_role_add_tag:nn {#1}{#2}
       }
  }

```

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

1.4.2 The pdf 2.0 version

__tag_role_add_tag:nnnn

The pdf 2.0 version takes four arguments: tag/namespace/role/namespace

```
\cs_new_protected:Nn \__tag_role_add_tag:nnnn %tag/namespace/role/namespace
  {
    \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
      {
        \msg_info:nnn { tag }{new-tag}{#1}
      }
    \__tag_seq_gput_right:Nn \g__tag_role_tags_seq { #1 }
    \__tag_prop_gput:Nnx \g__tag_role_tags_prop    { #1 }
```

```
      {
        \seq_count:N \g__tag_role_tags_seq
      }
    \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ #2 }
    \__tag_check_add_tag_role:nn {#1}{#3}
    \pdfdict_gput:nnx {g__tag_role/RoleMapNS_#2_dict}{#1}
      {
        [
          \pdf_name_from_unicode_e:n{#3}
          \c_space_tl
          \pdf_object_ref:n {tag/NS/#4}
        ]
      }
  }
\cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}
```

(*End definition for* `\__tag_role_add_tag:nnnn`.)

1.5 Key-val user interface

The user interface use the key `add-new-tag`, which takes either a keyval list as argument, or a tag/role.

`tag`
`tag-namespace`
`role`
`role-namespace`
`add-new-tag`

```
\keys_define:nn { __tag / tag-role }
  {
    ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
    ,tag-namespace  .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
    ,role .tl_set:N = \l__tag_role_role_tmpa_tl
    ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl
  }

\keys_define:nn { __tag / setup }
  {
    add-new-tag .code:n =
     {
       \keys_set_known:nnnN
         {__tag/tag-role}
         {
           tag-namespace=user,
           role-namespace=, %so that we can test for it.
          #1
         }{__tag/tag-role}\l_tmpa_tl
       \tl_if_empty:NF \l_tmpa_tl
         {
           \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/}
           \tl_set:Nx \l__tag_role_tag_tmpa_tl  { \seq_item:Nn \l_tmpa_seq {1} }
           \tl_set:Nx \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} }
         }
      \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
         {
           \prop_get:NVNTF
             \g__tag_role_tags_NS_prop
             \l__tag_role_role_tmpa_tl
```

```
              \l__tag_role_role_namespace_tmpa_tl
              {
                 \prop_if_in:NVF\g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
                  {
                    \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
                  }
              }
              {
                \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user}
              }
          }
       \pdf_version_compare:NnTF < {2.0}
        {
         %TODO add check for emptyness?
           \__tag_role_add_tag:VV
               \l__tag_role_tag_tmpa_tl
               \l__tag_role_role_tmpa_tl
        }
        {
          \__tag_role_add_tag:VVVV
            \l__tag_role_tag_tmpa_tl
            \l__tag_role_tag_namespace_tmpa_tl
            \l__tag_role_role_tmpa_tl
            \l__tag_role_role_namespace_tmpa_tl
        }
    }
  }
⟨/package⟩
```

(*End definition for* `tag` *and others. These functions are documented on page 62.*)

Part X

The **tagpdf-space** module Code related to real space chars Part of the tagpdf package

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-space-code} {2021-07-03} {0.91}
 {part of tagpdf - code related to real space chars}
⟨/header⟩
```

1 Code for interword spaces

The code is engine/backend dependant. Basically only pdftex and luatex support real space chars. Most of the code for luatex which uses attributes is in the lua code, here are only the keys.

interwordspace
show-spaces

```
⟨*package⟩
\sys_if_engine_pdftex:T
  {
    \sys_if_output_pdf:TF
      {
        \pdfglyphtounicode{space}{0020}
        \keys_define:nn { __tag / setup }
          {
            interwordspace .choices:nn = { true, on }  { \pdfinterwordspaceon },
            interwordspace .choices:nn = { false, off }{ \pdfinterwordspaceon },
            interwordspace .default:n = true,
            show-spaces .bool_set:N = \l__tag_showspaces_bool
          }
      }
      {
        \keys_define:nn { __tag / setup }
          {
            interwordspace .choices:nn = { true, on, false, off }
              { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi}  },
            interwordspace .default:n = true,
            show-spaces .bool_set:N = \l__tag_showspaces_bool
          }
      }
  }


\sys_if_engine_luatex:T
  {
    \keys_define:nn { __tag / setup }
      {
        interwordspace .choices:nn =
                                  { true, on }
```

```
                                {
                                  \bool_gset_true:N \g__tag_active_space_bool
                                  \lua_now:e{ltx.__tag.func.markspaceon()}
                                },
          interwordspace .choices:nn =
                                { false, off }
                                {
                                 \bool_gset_false:N \g__tag_active_space_bool
                                 \lua_now:e{ltx.__tag.func.markspaceoff()}
                                },
          interwordspace .default:n = true,
          show-spaces      .choice:,
          show-spaces  / true  .code:n =
                                {\lua_now:e{ltx.__tag.trace.showspaces=true}},
          show-spaces  / false .code:n =
                                {\lua_now:e{ltx.__tag.trace.showspaces=nil}},
          show-spaces .default:n = true
        }
  }

\sys_if_engine_xetex:T
  {
    \keys_define:nn { __tag / setup }
      {
        interwordspace .choices:nn = { true, on }
          { \msg_warning:nnn {tag}{sys-no-interwordspace}{xetex}  },
        interwordspace .choices:nn = { false, off }
          { \msg_warning:nnn {tag}{sys-no-interwordspace}{xetex}  },
        interwordspace .default:n = true,
        show-spaces .bool_set:N = \l__tag_showspaces_bool
      }
  }
```

(*End definition for* `interwordspace` *and* `show-spaces`*. These functions are documented on page* **??**.)

`\__tag_fakespace:` For luatex we need a command for the fake space as equivalent of the pdftex primitive.

```
\sys_if_engine_luatex:T
  {
    \cs_new_protected:Nn \__tag_fakespace:
      {
        \group_begin:
        \lua_now:e{ltx.__tag.func.fakespace()}
        \skip_horizontal:n{\c_zero_skip}
        \group_end:
      }
  }
⟨/package⟩
```

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

Index

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