tagpdf – LaTeX kernel code for PDF tagging[*]

Ulrike Fischer[†]

Released 2025-06-27

Contents

[*]This file describes v0.99s, last revised 2025-06-27.
[†]E-mail: fischer@troubleshooting-tex.de

Part I
The **tagpdf** main module Part of the tagpdf package

`\tag_suspend:n` `\tag_resume:n` `\tag_stop:n` `\tag_start:n`

```
\tag_suspend:n {⟨label⟩}
\tag_resume:n {⟨label⟩}
\tag_stop:n {⟨label⟩} (deprecated)
\tag_start:n {⟨label⟩} (deprecated)
```

We need commands to stop tagging in some places. They switches three local booleans and also stop the counting of paragraphs. If they are nested an inner `\tag_resume:n` will not restart tagging. ⟨`label`⟩ is only used in debugging messages to allow to follow the nesting and to identify which code is disabling the tagging. The label is not expanded so can be a single token, e.g. `\caption`. `\tag_suspend:n` and `\tag_resume:n` are the l3-layer variants of `\SuspendTagging` and `\ResumeTagging` and will be provided by the kernel in the next release.

`\tag_stop:` `\tag_start:` `\tagstop` `\tagstart`

deprecated These are variants of the above commands without the debuging level. They are now deprecated and it is recommended to use the kernel command `\SuspendTagging`, `\ResumeTagging`, `\tag_suspend:n` and `\tag_resume:n` instead.

`activate/spaces` (*setup key*)

`activate/spaces` activates the additional parsing needed for interword spaces. It replaces the deprecated key `interwordspace`.

`activate/mc` (*setup key*)
`vate-mc (deprecated)` (*setup key*)

A key to to activate the marked content code. It should be used only in special cases, e.g. for debugging.

`activate/tree` (*setup key*)
`te-tree (deprecated)` (*setup key*)

This key activates the code that finalize the various trees. It should be used only in special cases, e.g. for debugging.

`activate/struct` (*setup key*)
`-struct (deprecated)` (*setup key*)

This key activates the code for structures. It should be used only in special cases, e.g. for debugging.

`activate/all` (*setup key*)
`ate-all (deprecated)` (*setup key*)

This is a meta key for the three previous keys and is normally what should be used to activate tagging.

`activate/struct-dest` (*setup key*)
`ct-dest (deprecated)` (*setup key*)

The key allows to suppress the creation of structure destinations

`debug/log` (*setup key*)

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

`activate/tagunmarked` (*setup key*)
`nmarked (deprecated)` (*setup key*)

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

`activate/softhyphen` (*setup key*)

This key allows to activates automatic handling of hyphens inserted by hyphenation. It only is used in luamode and replaces hyphens by U+00AD if the font supports this.

page/tabsorder (*setup key*)
osorder (deprecated) (*setup key*)

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

tagstruct
tagstructobj
tagabspage
tagmcabs
tagmcid

These are attributes used by the label/ref system.

1 Initialization and test if pdfmanagement is active.

```
⟨@@=tag⟩
⟨*package⟩
\ProvidesExplPackage {tagpdf} {2025-06-27} {0.99s}
  { LaTeX kernel code for PDF tagging }

\IfPDFManagementActiveF
  {
    \PackageError{tagpdf}
     {
       PDF~resource~management~is~no~active!\MessageBreak
       tagpdf~will~no~work.
     }
     {
       Activate~it~with \MessageBreak
       \string\DocumentMetadata{<options>}\MessageBreak
       before~\string\documentclass
     }
  }
⟨/package⟩
```

<*debug>

```
\ProvidesExplPackage {tagpdf-debug} {2025-06-27} {0.99s}
  { debug code for tagpdf }
\@ifpackageloaded{tagpdf}{}{\PackageWarning{tagpdf-debug}{tagpdf~not~loaded,~quitting}\endinp
```

</debug> We map the internal module name “tag” to “tagpdf” in messages.

```
⟨*package⟩
\prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
⟨/package⟩
```

Debug mode has its special mapping:

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

2 base package

To avoid to have to test everywhere if tagpdf has been loaded and is active, we define a base package with dummy functions

```
⟨*base⟩
```

```
\ProvidesExplPackage {tagpdf-base} {2025-06-27} {0.99s}
  {part of tagpdf - provide base, no-op versions of the user commands }
⟨/base⟩
```

3 Package options

The boolean is kept for now for compatibility, can go in 2026.

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

4 Packages

To be on the safe side for now, load also the base definitions

```
\RequirePackage{tagpdf-base}
⟨/package⟩
```

The no-op version should behave a near enough to the real code as possible, so we define a command which a special in the relevant backends:

```
⟨*base⟩
\cs_new_protected:Npn \__tag_whatsits: {}
\AddToHook{begindocument}
 {
  \str_case:onF { \c_sys_backend_str }
   {
    { luatex  } { \cs_set_protected:Npn \__tag_whatsits: {} }
    { dvisvgm } { \cs_set_protected:Npn \__tag_whatsits: {} }
   }
   {
     \cs_set_protected:Npn \__tag_whatsits: {\tex_special:D {} }
   }
 }
⟨/base⟩
```

4.1 a LastPage label

With LaTeX 2025-06-01 we no longer need a special version as the label is now written directly.

```
⟨*package⟩
 \AddToHook{enddocument/afterlastpage}
  {\property_record:nn{@tag@LastPage}{abspage,tagmcabs,tagstruct}}
```

5 Variables

\l__tag_tmpa_tl
\l__tag_tmpb_tl
\l__tag_tmpc_tl
l__tag_tmp_unused_tl \l__tag_Ref_tmpa_tl
\l__tag_get_tmpc_tl
\l__tag_get_parent_tmpa_tl
\l__tag_get_parent_tmpb_tl
\l__tag_get_parent_tmpc_tl
\l__tag_get_child_tmpa_tl
\l__tag_get_child_tmpb_tl
\l__tag_get_child_tmpc_tl
\l__tag_tmpa_str
\l__tag_tmpa_prop

A few temporary variables

```
\tl_new:N    \l__tag_tmpa_tl
\tl_new:N    \l__tag_tmpb_tl
```

```
\tl_new:N    \l__tag_tmpc_tl
\tl_new:N    \l__tag_tmp_unused_tl
\tl_new:N    \l__tag_Ref_tmpa_tl
\tl_new:N    \l__tag_get_tmpc_tl
\tl_new:N    \l__tag_get_parent_tmpa_tl
\tl_new:N    \l__tag_get_parent_tmpb_tl
\tl_new:N    \l__tag_get_parent_tmpc_tl
\tl_new:N    \l__tag_get_child_tmpa_tl
\tl_new:N    \l__tag_get_child_tmpb_tl
\tl_new:N    \l__tag_get_child_tmpc_tl
\str_new:N   \l__tag_tmpa_str
\prop_new:N  \l__tag_tmpa_prop
\seq_new:N   \l__tag_tmpa_seq
\seq_new:N   \l__tag_tmpb_seq
\clist_new:N \l__tag_tmpa_clist
\int_new:N   \l__tag_tmpa_int
\box_new:N   \l__tag_tmpa_box
\box_new:N   \l__tag_tmpb_box
```

(*End of 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_property_mc_clist`
`\c__tag_property_struct_clist`

```
\clist_const:Nn \c__tag_property_mc_clist     {tagabspage,tagmcabs,tagmcid}
\clist_const:Nn \c__tag_property_struct_clist {tagstruct,tagstructobj}
```

(*End of definition for* `\c__tag_property_mc_clist` *and* `\c__tag_property_struct_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 of definition for* `\l__tag_loglevel_int`*.*)

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

These booleans should help to control the global behaviour of tagpdf. Ideally it should more or less do nothing if all are false. The space-boolean controls the interword space code, the mc-boolean activates `\tag_mc_begin:n`, the tree-boolean activates writing the finish code and the pdfmanagement related commands, the struct-boolean activates the storing of the structure data. In a normal document all should be active, the split is only there for debugging purpose. Structure destination will be activated automatically, but with the boolean struct-dest-boolean one can suppress them. Also we assume currently that they are set only at begin document. But if some control passing over groups are needed they could be perhaps used in a document too. TODO: check if they are used everywhere as needed and as wanted.

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

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

\l__tag_active_mc_bool
\l__tag_active_struct_bool
\l__tag_active_socket_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
\bool_new:N \l__tag_active_socket_bool
```

(*End of definition for* \l__tag_active_mc_bool *,* \l__tag_active_struct_bool *, and* \l__tag_active_-socket_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 of definition for* \g__tag_tagunmarked_bool*.*)

\g__tag_softhyphen_bool

This boolean controls if the code should try to automatically handle hyphens from hyphenation. It is currently only used in luamode.

```
\bool_new:N \g__tag_softhyphen_bool
```

(*End of definition for* \g__tag_softhyphen_bool*.*)

\g__tag_unique_cnt_int

If tagpdf has to create unique names (e.g. for object names when embedding files) it can use this integer to get an unique name. At every use it should be increased

```
\int_new:N \g__tag_unique_cnt_int
```

(*End of definition for* \g__tag_unique_cnt_int*.*)

6 Variants of l3 commands

```
\prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F,TF}
\cs_generate_variant:Nn \pdf_object_ref:n {e}
\cs_generate_variant:Nn \pdfannot_dict_put:nnn {nne}
\cs_generate_variant:Nn \pdffile_embed_stream:nnn {nee,oee}
\cs_generate_variant:Nn \prop_gput:Nnn {Nee,Nen} %** unneeded
\cs_generate_variant:Nn \prop_put:Nnn  {Nee}     %** unneeded
\cs_generate_variant:Nn \prop_item:Nn {No,Ne}    %**  unneeded
\cs_generate_variant:Nn \seq_set_split:Nnn{Nno}
\cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
\cs_generate_variant:Nn \clist_map_inline:nn {on}
\cs_generate_variant:Nn \pdffile_embed_file:nnn {eee}
```

7 Label and Reference commands

The code uses mostly the kernel properties but need a few local variants.

__tag_property_record:nn The command to record a property while preserving the spaces similar to the standard `\label`.

```
    \cs_new_protected:Npn \__tag_property_record:nn #1#2
      {
        \@bsphack
        \property_record:nn{#1}{#2}
        \@esphack
      }

```

And a few variants

```
\cs_generate_variant:Nn \property_ref:nnn {enn}
\cs_generate_variant:Nn \property_ref:nn  {en}
\cs_generate_variant:Nn \__tag_property_record:nn {en,eo}
```

(*End of definition for* `\__tag_property_record:nn`*.*)

__tag_property_ref_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_property_ref_lastpage:nn #1 #2
  {
    \property_ref:nnn {@tag@LastPage}{#1}{#2}
  }
```

(*End of definition for* `\__tag_property_ref_lastpage:nn`*.*)

8 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`.

```
\property_new:nnnn
  { tagstruct } { now }
  {1} { \int_use:N \c@g__tag_struct_abs_int }
\property_new:nnnn  { tagstructobj } { now }  {}
  {
     \pdf_object_ref_indexed:nn { __tag/struct } { \c@g__tag_struct_abs_int }
  }
\property_new:nnnn
  { tagabspage } { shipout }
  {0} { \int_use:N \g_shipout_readonly_int }
\property_new:nnnn  { tagmcabs } { now }
  {0} { \int_use:N \c@g__tag_MCID_abs_int }

\flag_new:n { __tag/mcid }
\property_new:nnnn  {tagmcid } { shipout }
   {0} { \flag_height:n { __tag/mcid } }

```

(*End of definition for* `tagstruct` *and others. These functions are documented on page 8.*)

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_prop_new_linked: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_prop_new_linked:N \prop_new_linked: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_gput_left:Nn  \seq_gput_left: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
% cnx temporary needed for latex-lab-graphic code
\cs_generate_variant:Nn \__tag_prop_gput:Nnn      { Nen, Nee, Nne, Nno, cnn, cen, cne, cno, c
\cs_generate_variant:Nn \__tag_seq_gput_right:Nn  { Ne  , No, cn, ce }
\cs_generate_variant:Nn \__tag_seq_gput_left:Nn   { ce }
\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 }
⟨/package⟩
```

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

10 General tagging commands

\tag_suspend:n
\tag_resume:n
\tag_stop:
\tag_start:
\tag_stop:n
\tag_start:n

We need commands to stop tagging in some places. They switch local booleans and also stop the counting of paragraphs. The commands keep track of the nesting with a local counter. Tagging only is only restarted at the outer level, if the current level is 1. The commands with argument allow to give a label. This is only used in debugging messages to allow to follow the nesting. The label is not expand so can e.g. be a single command token.

When stop/start pairs are nested we do not want the inner start command to restart tagging. To control this we use a local int: The stop command will increase it. The starting will decrease it and only restart tagging, if it is zero. This will replace the label version.

```
⟨*package | debug⟩
⟨package⟩\int_new:N \l__tag_tag_stop_int
```

\l__tag_tag_stop_int

```
\cs_set_protected:Npn \tag_stop:
  {
⟨debug⟩      \msg_note:nne {tag / debug }{tag-suspend}{ \int_use:N \l__tag_tag_stop_int }
    \int_incr:N \l__tag_tag_stop_int
    \bool_set_false:N \l__tag_active_struct_bool
    \bool_set_false:N \l__tag_active_mc_bool
    \bool_set_false:N \l__tag_active_socket_bool
    \__tag_stop_para_ints:
  }
```

```
\cs_set_protected:Npn \tag_start:
  {
    \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
    \int_if_zero:nT { \l__tag_tag_stop_int }
      {
        \bool_set_true:N \l__tag_active_struct_bool
        \bool_set_true:N \l__tag_active_mc_bool
        \bool_set_true:N \l__tag_active_socket_bool
        \__tag_start_para_ints:
      }
⟨debug⟩    \msg_note:nne {tag / debug }{tag-resume}{ \int_use:N \l__tag_tag_stop_int }
  }
\cs_set_eq:NN\tagstop\tag_stop:
\cs_set_eq:NN\tagstart\tag_start:

\cs_set_protected:Npn \tag_suspend:n #1
  {
⟨debug⟩    \msg_note:nnee {tag / debug }{tag-suspend}
⟨debug⟩       { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
    \int_incr:N \l__tag_tag_stop_int
    \bool_set_false:N \l__tag_active_struct_bool
    \bool_set_false:N \l__tag_active_mc_bool
    \bool_set_false:N \l__tag_active_socket_bool
    \__tag_stop_para_ints:
  }
\cs_set_eq:NN \tag_stop:n \tag_suspend:n
\cs_set_protected:Npn \tag_resume:n #1
  {
    \int_if_zero:nF { \l__tag_tag_stop_int } { \int_decr:N \l__tag_tag_stop_int }
    \int_if_zero:nT { \l__tag_tag_stop_int }
      {
        \bool_set_true:N \l__tag_active_struct_bool
        \bool_set_true:N \l__tag_active_mc_bool
        \bool_set_true:N \l__tag_active_socket_bool
        \__tag_start_para_ints:
      }
⟨debug⟩    \msg_note:nnee {tag / debug }{tag-resume}
⟨debug⟩       { \int_use:N \l__tag_tag_stop_int }{\exp_not:n{#1}}
  }
\cs_set_eq:NN \tag_start:n \tag_resume:n
⟨/package | debug⟩
⟨*base⟩
\cs_new_protected:Npn \tag_stop:{}
\cs_new_protected:Npn \tag_start:{}
\cs_new_protected:Npn \tagstop{}
\cs_new_protected:Npn \tagstart{}
\cs_new_protected:Npn \tag_stop:n  #1 {}
\cs_new_protected:Npn \tag_start:n #1 {}
```

Until the commands are provided by the kernel we provide them here too

```
\cs_set_eq:NN \tag_suspend:n \tag_stop:n
\cs_set_eq:NN \tag_resume:n  \tag_start:n
⟨/base⟩
```

(*End of definition for* `\tag_suspend:n` *and others. These functions are documented on page 7.*)

11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

activate/mc (*setup key*)
activate/tree (*setup key*)
activate/struct (*setup key*)
activate/all (*setup key*)
ctivate/struct-dest (*setup key*)

Keys to (globally) activate tagging. `activate/spaces` activates the additional parsing needed for interword spaces. It is defined in tagpdf-space. `activate/struct-dest` allows to activate or suppress structure destinations.

```
⟨*package⟩
\keys_define:nn { __tag / setup }
  {
    activate/mc     .bool_gset:N = \g__tag_active_mc_bool,
    activate/tree   .bool_gset:N = \g__tag_active_tree_bool,
    activate/struct .bool_gset:N = \g__tag_active_struct_bool,
    activate/all    .meta:n =
      {activate/mc={#1},activate/tree={#1},activate/struct={#1}},
    activate/all  .default:n = true,
    activate/struct-dest .bool_gset:N = \g__tag_active_struct_dest_bool,
```

old, deprecated names

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

debug/show (*setup key*)

Subkeys/values are defined in various other places.

```
    debug/show            .choice:,
```

debug/log (*setup key*)
debug/uncompress (*setup key*)
log (deprecated) (*setup key*)
ompress (deprecated) (*setup key*)

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

```
    debug/log             .choice:,
    debug/log / none      .code:n = {\int_set:Nn \l__tag_loglevel_int { 0 }},
    debug/log / v         .code:n =
      {
        \int_set:Nn \l__tag_loglevel_int { 1 }
        \cs_set_protected:Nn \__tag_check_typeout_v:n { \iow_term:e {##1} }
      },
    debug/log / vv        .code:n = {\int_set:Nn \l__tag_loglevel_int { 2 }},
    debug/log / vvv       .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
    debug/log / all       .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
    debug/uncompress .code:n = { \pdf_uncompress:  },
```

deprecated but still needed as the documentmetadata key argument uses it.

```
    log             .meta:n = {debug/log={#1}},
    uncompress      .code:n = { \pdf_uncompress:  },
```

ctivate/tagunmarked (*setup key*)
nmarked (deprecated) (*setup key*)

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

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

deprecated name

```
    tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
```

`activate/softhyphen` (*setup key*) This key activates (in luamode) the handling of soft hyphens.

```
    activate/softhyphen     .bool_gset:N = \g__tag_softhyphen_bool,
    activate/softhyphen     .initial:n  = true,
```

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

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

deprecated name

```
    tabsorder .meta:n = {page/tabsorder={#1}},
  }
```

12 loading of engine/more dependent code

```
\sys_if_engine_luatex:T
  {
    \file_input:n {tagpdf-luatex.def}
  }
⟨/package⟩
```

```
⟨*mcloading⟩
\bool_if:NTF \g__tag_mode_lua_bool
  {
   \RequirePackage {tagpdf-mc-code-lua}
  }
  {
   \RequirePackage {tagpdf-mc-code-generic} %
  }
⟨/mcloading⟩
⟨*debug⟩
\bool_if:NTF \g__tag_mode_lua_bool
  {
   \RequirePackage {tagpdf-debug-lua}
  }
  {
   \RequirePackage {tagpdf-debug-generic} %
  }
⟨/debug⟩
```

Part II

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 for the current structure or mc-chunk. Currently the only sensible values for the argument ⟨*keyword*⟩ are `mc_tag`, `struct_tag`, `struct_id` and `struct_num`.

`\tag_if_box_tagged_p:N ★`
`\tag_if_box_tagged:NTF ★`

`\tag_if_box_tagged:NTF ⟨box⟩ {⟨true code⟩} {⟨false code⟩}`

This tests if a box contains tagging commands. It relies currently on that the code, that saved the box, correctly sets the command `\l_tag_box_\int_use:N #1_tl` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

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	
`\ShowTagging`debug/structures = num	log+termn	debug mode only

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
`\@@_struct_write_obj:n`	`struct-orphan`	warning
`\tag_struct_begin:n`	`struct-faulty-nesting`	error
`\@@_struct_insert_annot:nn`	`struct-faulty-nesting`	error
`tag_struct_use:n`	`struct-label-unknown`	warning
`attribute-class`, `attribute`	`attr-unknown`	error
`\@@_tree_fill_parenttree:`	`tree-mcid-index-wrong`	warning TODO: should trigger a standard rerun m
in `enddocument/info-hook`	`para-hook-count-wrong`	error (warning?)

2.3 Messages from the ptagging code

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

2.4 Warning messages from the lua-code

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

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

2.5 Info messages from the lua-code

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

`message`	log-level	remark
`INFO SHIPOUT-INSERT-LAST-EMC`	3	finish of shipout code
`INFO SPACE-FUNCTION-FONT`	3	interwordspace code
`INFO TAG-ABSPAGE`	3	
`INFO TAG-ARGS`	4	
`INFO TAG-ENDHEAD`	4	
`INFO TAG-ENDHEAD`	4	
`INFO TAG-HEAD`	3	
`INFO TAG-INSERT-ARTIFACT`	3	

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

2.6 Debug mode messages and code

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

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

2.7 Messages

`mc-nested`
`mc-tag-missing`
`mc-label-unknown`
`mc-used-twice`
`mc-not-open`
`mc-pushed`
`mc-popped`
`mc-current`

Various messages related to mc-chunks. TODO document their meaning.

`struct-unknown`
`struct-no-objnum`
`struct-orphan`
`struct-faulty-nesting`
`struct-missing-tag`
`struct-used-twice`
`struct-label-unknown`
`struct-show-closing`

Various messages related to structure. Check the definition in the code for their meaning and the arguments they take.

`tree-struct-still-open` Message issued at the end of the compilation if there are (beside Root) other open structures on the stack.

`tree-statistic` Message issued at the end of the compilation showing the number of objects to write

`show-struct`
`show-kids`

These two messages are used in debug mode to show the current structures in the log and terminal.

`attr-unknown` Message if an attribute i sunknown.

`role-missing`
`role-unknown`
`role-unknown-tag`
`role-unknown-NS`
`role-tag`
`new-tag`
`role-parent-child-result`
`role-remapping`

Messages related to role mapping.

`tree-mcid-index-wrong` Used in the tree code, typically indicates the document must be rerun.

`sys-no-interwordspace` Message if an engine doesn't support inter word spaces

`para-hook-count-wrong` Message if the number of begin paragraph and end paragraph differ. This normally means faulty structure.

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-checks-code} {2025-06-27} {0.99s}
 {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 of definition for* `mc-nested`. *This function is documented on page 19.*)

mc-tag-missing If the tag is missing

```
\msg_new:nnn { tag } {mc-tag-missing} { MC-tag~missing;~#1~used~instead~-~mcid~#2 }
```

(*End of definition for* `mc-tag-missing`. *This function is documented on page 19.*)

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 of definition for* `mc-label-unknown`. *This function is documented on page 19.*)

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 of definition for* `mc-used-twice`. *This function is documented on page 19.*)

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 of definition for* `mc-not-open`. *This function is documented on page 19.*)

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 of definition for* `mc-pushed` *and* `mc-popped`. *These functions are documented on page 19.*)

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 of definition for* `mc-current`. *This function is documented on page 19.*)

3.2 Messages related to structures

struct-unknown if for example a parent key value points to structure that doesn't exist (yet)

```
\msg_new:nnn { tag } {struct-unknown}
  { structure~with~number~#1~doesn't~exist\\ #2 }
```

(*End of definition for* `struct-unknown`. *This function is documented on page 20.*)

struct-no-objnum Should not happen ...

```
\msg_new:nnn { tag } {struct-no-objnum} { objnum~missing~for~structure~#1 }
```

(*End of definition for* `struct-no-objnum`. *This function is documented on page 20.*)

struct-orphan This indicates that there is a structure which has kids but no parent. This can happen if a structure is stashed but then not used.

```
\msg_new:nnn { tag } {struct-orphan}
  {
    Structure~#1~has~#2~kids~but~no~parent.\\
    It~is~turned~into~an~artifact.\\
    Did~you~stashed~a~structure~and~then~didn't~use~it?
  }

```

(*End of definition for* `struct-orphan`. *This function is documented on page 20.*)

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 of definition for* `struct-faulty-nesting`. *This function is documented on page 20.*)

struct-missing-tag A structure must have a tag.

```
\msg_new:nnn { tag } {struct-missing-tag} { a~structure~must~have~a~tag! }
```

(*End of definition for* `struct-missing-tag`. *This function is documented on page 20.*)

struct-used-twice

```
\msg_new:nnn { tag } {struct-used-twice}
  { structure~with~label~#1~has~already~been~used}
```

(*End of definition for* `struct-used-twice`. *This function is documented on page 20.*)

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 of definition for* `struct-label-unknown`. *This function is documented on page 20.*)

struct-show-closing Informational message shown if log-mode is high enough

```
\msg_new:nnn { tag } {struct-show-closing}
  { closing~structure~#1~tagged~\use:e{\prop_item:cn{g__tag_struct_#1_prop}{S}} }
```

(*End of definition for* `struct-show-closing`. *This function is documented on page 20.*)

`struct-Ref-unknown` This message is issued at the end, when the Ref keys are updated. TODO: in debug mode it should report more info about the structure.

```
\msg_new:nnn { tag } {struct-Ref-unknown}
 {
     #1~has~no~related~structure.\\
    /Ref~not~updated.
 }
```

(*End of definition for* `struct-Ref-unknown`. *This function is documented on page* **??**.)

`tree-struct-still-open` Message issued at the end if there are beside Root other open structures on the stack.

```
\msg_new:nnn { tag } {tree-struct-still-open}
  {
    There~are~still~open~structures~on~the~stack!\\
    The~stack~contains~\seq_use:Nn\g__tag_struct_tag_stack_seq{,}.\\
    The~structures~are~automatically~closed,\\
    but~their~nesting~can~be~wrong.
  }
```

(*End of definition for* `tree-struct-still-open`. *This function is documented on page 20.*)

`tree-statistic` Message issued at the end showing the estimated number of structures and MC-childs

```
\msg_new:nnn { tag } {tree-statistic}
  {
    Finalizing~the~tagging~structure:\\
    Writing~out~\c_tilde_str
    \int_use:N\c@g__tag_struct_abs_int\c_space_tl~structure~objects\\
    with~\c_tilde_str
    \int_use:N\c@g__tag_MCID_abs_int\c_space_tl'MC'~leaf~nodes.\\
    Be~patient~if~there~are~lots~of~objects!
  }
⟨/package⟩
```

(*End of definition for* `tree-statistic`. *This function is documented on page 20.*)

The following messages are only needed in debug mode.

`show-struct`
`show-kids` This two messages are used to show the current structures in the log and terminal.

```
⟨*debug⟩
\msg_new:nnn { tag/debug } { show-struct }
  {
    =========================\\
    The~structure~#1~
    \tl_if_empty:nTF {#2}
      { is~empty \\>~ . }
      { contains: #2  }
    \\
  }
\msg_new:nnn { tag/debug } { show-kids }
  {
    The~structure~has~the~following~kids:
    \tl_if_empty:nTF {#2}
      { \\>~ NONE }
      { #2  }
    \\
      =========================
  }
⟨/debug⟩
```

(*End of definition for* `show-struct` *and* `show-kids`*. These functions are documented on page 20.*)

3.3 Attributes

Not much yet, as attributes aren't used so much.

`attr-unknown`

```
⟨*package⟩
\msg_new:nnn { tag } {attr-unknown}  { attribute~#1~is~unknown}
```

(*End of definition for* `attr-unknown`*. This function is documented on page 20.*)

3.4 Roles

`role-missing`
`role-unknown`
`role-unknown-tag`
`role-unknown-NS`

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  }
\msg_new:nnn { tag } {role-unknown-NS}  { \tl_if_empty:nTF{#1}{Empty~NS}{NS~#1~is~not~known}
```

(*End of definition for* `role-missing` *and others. These functions are documented on page 20.*)

`role-parent-child-check`

This is an info message that inform which elements are checked, typically used to show the original tags, not the rolemapped one.

```
\msg_new:nnn { tag } {role-parent-child-check}
  { Checking~Parent-Child~'#1'~-->~'#2' }
```

(*End of definition for* `role-parent-child-check`*. This function is documented on page* **??**.)

`role-parent-child-result`

This is info and warning message about the containment rules between child and parent tags.

```
\msg_new:nnn { tag } {role-parent-child-result}
  { Parent-Child~'#1'~-->~'#2'.\\Relation~is~#3~\msg_line_context:}
```

(*End of definition for* `role-parent-child-result`*. This function is documented on page 20.*)

`role-struct-parent-child-forbidden`

The most important message is that the relation is not allowed between two structures. Argument #1 is the parent structure number, #2 is the child structure number, #3 NS:tag info of the parent (TODO perhaps rolemapped), #4 NS:tag of the child. (TODO)

```
\msg_new:nnn { tag } {role-struct-parent-child-forbidden}
  {
    Parent-Child~'#3'~-->~'#4'.\\
    Relation~is~not~allowed! ~\msg_line_context:\\
    struct~#1,~
    \exp_last_unbraced:Ne\use_i:nn  { \prop_item:cn{ g__tag_struct_#1_prop}{tag} }
    \c_space_tl-->\c_space_tl
    struct~#2,~
    \exp_last_unbraced:Ne\use_i:nn  { \prop_item:cn{ g__tag_struct_#2_prop}{tag} }
  }
```

(*End of definition for* `role-struct-parent-child-forbidden`*. This function is documented on page* **??**.)

`role-MC-child-forbidden` In case that MC is forbidden we use a special message. Argument #1 is the parent structure number. #2 NS:tag of the parent,

```
\msg_new:nnn { tag } {role-MC-child-forbidden}
  {
    Parent-Child~'#2'~-->~'MC~(real~content)'.\\
    Relation~is~not~allowed! ~\msg_line_context:\\
    struct~#1,~
    \exp_last_unbraced:Ne\use_i:nn { \prop_item:cn{ g__tag_struct_#1_prop}{tag} }
  }
```

(*End of definition for* `role-MC-child-forbidden`*. This function is documented on page* **??**.)

`role-parent-child-forbidden` The most important message is that the relation is not allowed. Argument #1 is the parent structure number. #2 NS:tag of the parent, #3 NS:tag of the child.

```
\msg_new:nnn { tag } {role-parent-child-forbidden}
  {
    Parent-Child~'#2'~-->~'#3'.\\
    Relation~is~not~allowed! ~\msg_line_context:\\
    struct~#1,~\prop_item:cn{ g__tag_struct_#1_prop}{S}
    \c_space_tl
    \str_if_eq:nnF{#3}{MC~(realcontent)}
      {-->~struct~\int_eval:n {\c@g__tag_struct_abs_int}}
  }
```

(*End of definition for* `role-parent-child-forbidden`*. This function is documented on page* **??**.)

`\__tag_check_forbidden_parent_child:nnnn`

```
\cs_new_protected:Npn \__tag_check_forbidden_parent_child:nnnn #1#2#3#4
% #1 check number, #2 number of parent struct
% #3 parent info,  #4 child info
 {
   \int_compare:nNnT {#1 } <0
     {
       \msg_warning:nneee
         { tag }
         {role-parent-child-forbidden}
         { #2}
         { #3 }
         { #4 }
     }
 }
\cs_generate_variant:Nn \__tag_check_forbidden_parent_child:nnnn {nnee}

% new with structure numbers:
\cs_new_protected:Npn \__tag_check_struct_forbidden_parent_child:nnn #1#2#3
% #1 check number,
% #2 number of parent struct
% #3 number of child struct
 {
   \int_compare:nNnT {#1 } <0
     {
       \prop_get:cnN {g__tag_struct_#2_prop}{parentrole}\l__tag_get_parent_tmpc_tl
```

```
        \prop_get:cnN {g__tag_struct_#3_prop}{rolemap}\l__tag_get_child_tmpc_tl
        \msg_warning:nneeee
          { tag }
          {role-struct-parent-child-forbidden}
          { #2 }
          { #3 }
          {
            \exp_last_unbraced:No \use_ii:nn  { \l__tag_get_parent_tmpc_tl }
            :
            \exp_last_unbraced:No \use_i:nn  {\l__tag_get_parent_tmpc_tl }
          }
          {
            \exp_last_unbraced:No \use_ii:nn { \l__tag_get_child_tmpc_tl }
            :
            \exp_last_unbraced:No \use_i:nn { \l__tag_get_child_tmpc_tl }
          }
      }
  }
\cs_generate_variant:Nn\__tag_check_struct_forbidden_parent_child:nnn{onn}
```

(*End of definition for* `\__tag_check_forbidden_parent_child:nnnn`*.*)

`role-parent-child-unresolved` If a structure is stashed and then used later and its root is one of Part, Div or NonStruct, then we can not check the parent-child rules. This would require to know all children. In this case we only warn. resolved a Argument #1 is the parent structure number. #2 NS:tag of the parent, #3 NS:tag of the child.

```
\msg_new:nnn { tag } {role-parent-child-unresolved}
  {
    Structure~\int_eval:n {\c@g__tag_struct_abs_int}~was~moved~into~structure~#1.\\
    Parent-Child~'#2'~-->~'#3'~can~not~checked.
  }
```

(*End of definition for* `role-parent-child-unresolved`*. This function is documented on page* **??***.*)

`\__tag_check_unresolved_parent_child:nnnn`

```
\cs_new_protected:Npn \__tag_check_unresolved_parent_child:nnnn #1#2#3#4
% #1 check number, #2 number of parent struct
% #3 parent info, #4 child info
 {
   \int_compare:nNnT { #1 } = {\c__tag_role_rule_checkparent_tl}
     {
       \msg_warning:nneee
         { tag }
         {role-parent-child-unresolved}
         { #2 }
         { #3 }
         { #4 }
     }
 }
```

(*End of definition for* `\__tag_check_unresolved_parent_child:nnnn`*.*)

`tag/check/parent-child`
`tag/check/parent-child-end` Sockets used around the parent-child checks so that we can disable them.

```
\socket_new:nn{tag/check/parent-child}{1}
```

```
\socket_new:nn{tag/check/parent-child-end}{0}
\socket_new_plug:nnn {tag/check/parent-child-end}{check}
  {
   \sys_if_engine_luatex:T
     {
       \lua_now:e
        {
          ltx.__tag.func.check_parent_child_rules ( 2 )
        }
     }
  }
```

And a key to disable the check

```
\keys_define:nn { __tag / setup}
  {
    debug / parent-child-check .choice:,
    debug / parent-child-check / on .code:n =
     {
       \socket_assign_plug:nn {tag/check/parent-child}{identity}
     },
    debug / parent-child-check / off .code:n=
     {
       \socket_assign_plug:nn {tag/check/parent-child}{noop}
       \socket_assign_plug:nn {tag/check/parent-child-end}{noop}
     },
    debug / parent-child-check / atend .code:n=
     {
       \socket_assign_plug:nn {tag/check/parent-child}{noop}
       \socket_assign_plug:nn {tag/check/parent-child-end}{check}
     }
  }
```

(*End of definition for* `tag/check/parent-child` *and* `tag/check/parent-child-end`*. These functions are documented on page* **??**.)

`role-remapping` This is info and warning message about role-remapping

```
\msg_new:nnn { tag } {role-remapping}
  { remapping~tag~to~#1 }
```

(*End of definition for* `role-remapping`*. This function is documented on page 20.*)

`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 }
\msg_new:nnn { tag } {read-namespace}    { reading~namespace~definitions~tagpdf-
ns-#1.def }
\msg_new:nnn { tag } {namespace-missing}{ namespace~definitions~tagpdf-ns-#1.def~not~found }
\msg_new:nnn { tag } {namespace-unknown}{ namespace~#1~is~not~declared }
```

(*End of definition for* `role-tag` *and* `new-tag`*. These functions are documented on page 20.*)

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 of definition for* `tree-mcid-index-wrong`. *This function is documented on page 20.*)

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 of definition for* `sys-no-interwordspace`. *This function is documented on page 20.*)

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

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

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

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

```
\msg_new:nnnn { tag } {para-hook-count-wrong}
  {The~number~of~automatic~begin~(#1)~and~end~(#2)~#3~para~hooks~differ!}
  {This~quite~probably~a~coding~error~and~the~structure~will~be~wrong!}
⟨/package⟩
```

(*End of definition for* `para-hook-count-wrong`. *This function is documented on page 20.*)

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`, `struct_tag` and `struct_num`.

```
⟨base⟩\cs_new:Npn \tag_get:n #1   { \use:c {__tag_get_data_#1: } }
```

(*End of definition for* `\tag_get:n`. *This function is documented on page 17.*)

5 User conditionals

\tag_if_active_p:
\tag_if_active:*TF* This tests if 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.

```
⟨*base⟩
\cs_if_exist:NF\tag_if_active:T
 {
   \prg_new_conditional:Npnn \tag_if_active: { p , T , TF, F }
     { \prg_return_false: }
 }
⟨/base⟩
⟨*package⟩
\prg_set_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:
      }
  }
⟨/package⟩
```

(*End of definition for* `\tag_if_active:TF`. *This function is documented on page 17.*)

`\tag_if_box_tagged_p:N`
`\tag_if_box_tagged:NTF`

This tests if a box contains tagging commands. It relies on that the code that saved the box correctly set `\l_tag_box_<box number>_tl` to a positive value. The LaTeX commands will do that automatically at some time but it is in the responsibility of the user to ensure that when using low-level code. If the internal command doesn't exist the box is assumed to be untagged.

```
⟨*base⟩
\prg_new_conditional:Npnn \tag_if_box_tagged:N #1 {p,T,F,TF}
    {
      \tl_if_exist:cTF {l_tag_box_\int_use:N #1_tl}
      {
        \int_compare:nNnTF {0\tl_use:c{l_tag_box_\int_use:N #1_tl}}>{0}
         { \prg_return_true:  }
         { \prg_return_false: }
      }
      {
        \prg_return_false:
        % warning??
      }
    }
⟨/base⟩
```

(*End of definition for* `\tag_if_box_tagged:NTF`. *This function is documented on page 17.*)

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`

This checks if mc are active.

```
⟨*package⟩
\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 of definition for* `\__tag_check_if_active_mc:TF` *and* `\__tag_check_if_active_struct:TF`*.*)

6.2 Checks related to structures

`\__tag_check_structure_has_tag:n`

Structures must have a tag, so we check if the S entry is in the property. It is an error if this is missing. The argument is a number. 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_get:cnNF
      { g__tag_struct_#1_prop }
      {S}
      \l__tag_tmp_unused_tl
      {
        \msg_error:nn { tag } {struct-missing-tag}
      }
  }
```

(*End of 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_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
      {
        \msg_warning:nne { tag } {role-unknown-tag} {#1}
      }
  }
```

(*End of 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,e}
```

(*End of 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 of 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_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop}
      {parentnum}
      \l__tag_tmpa_tl
      {
        \msg_warning:nnn { tag } {struct-used-twice} {#1}
      }
  }
```

(*End of 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_error:nnn { tag } {role-missing} {#1}
      }
      {
        \prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l__tag_tmpa_tl
          {
            \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
              {
                \msg_info:nnnn { tag } {role-tag} {#1} {#2}
              }
          }
          {
            \msg_error:nnn { tag } {role-unknown} {#2}
          }
      }
  }
```

Similar with a namespace

```
\cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
  {
    \tl_if_empty:nTF {#2}
      {
        \msg_error:nnn { tag } {role-missing} {#1}
      }
      {
        \prop_get:cnNTF { g__tag_role_NS_#3_prop } {#2} \l__tag_tmpa_tl
          {
            \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
              {
                \msg_info:nnnn { tag } {role-tag} {#1} {#2/#3}
              }
          }
          {
            \msg_error:nnn { tag } {role-unknown} {#2/#3}
          }
      }
  }
```

(*End of 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:nne { tag } {mc-nested} { \__tag_get_mc_abs_cnt: }
      }
  }

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

(*End of 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:nne {tag}{mc-#1}{#2} }
    \int_compare:nNnT
```

```
      { \l__tag_loglevel_int } > { 2 }
      {
        \msg_info:nne {tag}{mc-#1}{#2}
        \seq_log:N \g__tag_mc_stack_seq
      }
  }
```

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

`\__tag_check_mc_tag:N`

This checks if the mc has a (known) tag, if it is empty (e.g. if due to a call to the output routine, see issue https://github.com/latex3/tagpdf/issues/111) then we fall back to the structure name.

```
\cs_new_protected:Npn \__tag_check_mc_tag:N #1  %#1 is var with a tag name in it
  {
    \tl_if_empty:NTF #1
      {
        \tl_set:No #1 { \g__tag_struct_tag_tl }
        \msg_info:nnee { tag } {mc-tag-missing} { \g__tag_struct_tag_tl }{ \__tag_get_mc_abs_
      }
      {
       \prop_get:NoNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
         {
           \msg_warning:nne { tag } {role-unknown-tag} {#1}
         }
      }
  }
```

(*End of 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 of 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 of 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:Ne \l__tag_tmpa_tl
      {
        \__tag_property_ref_lastpage:nn
          {abspage}
          {-1}
      }
    \int_step_inline:nnnn {1}{1}
      {
        \l__tag_tmpa_tl
      }
      {
        \seq_clear:N \l__tag_tmpa_seq
        \int_step_inline:nnnn
          {1}
          {1}
          {
            \__tag_property_ref_lastpage:nn
              {tagmcabs}
              {-1}
          }
          {
            \int_compare:nT
              {
                \property_ref:enn
                  {mcid-####1}
                  {tagabspage}
                  {-1}
                =
                ##1
             }
             {
               \seq_gput_right:Ne \l__tag_tmpa_seq
                 {
                   Page##1-####1-
                   \property_ref:enn
                     {mcid-####1}
                     {tagmcid}
                     {-1}
```

```
                    }
                }
            }
            \seq_show:N \l__tag_tmpa_seq
        }
    }
```

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

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

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

`\__tag_check_mc_in_galley_p:`
`\__tag_check_mc_in_galley:TF`

At first we need a test to decide if `\tag_mc_begin:n` (tmb) and `\tag_mc_end:` (tme) has been used at all on the current galley. As each command issues two slightly different marks we can do it by comparing firstmarks and botmarks. The test assumes that the marks have been already mapped into the sequence with `\@@_mc_get_marks:`. As `\seq_if_eq:NNTF` doesn't exist we use the tl-test.

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

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

`\__tag_check_if_mc_tmb_missing_p:`
`\__tag_check_if_mc_tmb_missing:TF`

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

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

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

`\__tag_check_if_mc_tme_missing_p:`
`\__tag_check_if_mc_tme_missing:TF`

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

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

```
    { \prg_return_true: }
    { \prg_return_false: }
 }
```

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

```
⟨/package⟩
⟨*debug⟩
```

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

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

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

And now something for the structures

```
\msg_new:nnn { tag / debug } {struct-begin}
  {
    Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\\[\msg_line_contex
  }
\msg_new:nnn { tag / debug } {struct-end}
  {
    Struct~end~#1~[\msg_line_context:]
  }
\msg_new:nnn { tag / debug } {struct-end-wrong}
  {
    Struct~end~'#1'~doesn't~fit~start~'#2'~[\msg_line_context:]
  }

\cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
        \seq_log:N \g__tag_struct_tag_stack_seq
     }
 }
\cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnnn { tag / debug } {struct-begin } {ignored} { #1 }
     }
 }
\cs_new_protected:Npn \__tag_debug_struct_end_insert:
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnn { tag / debug } {struct-end} {inserted}
        \seq_log:N \g__tag_struct_tag_stack_seq
     }
 }
\cs_new_protected:Npn \__tag_debug_struct_end_ignore:
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
     {
        \msg_note:nnn { tag / debug } {struct-end } {ignored}
     }
 }
\cs_new_protected:Npn \__tag_debug_struct_end_check:n #1
 {
   \int_compare:nNnT { \l__tag_loglevel_int } > {0}
    {
      \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
        {
          \str_if_eq:eeF
           {#1}
           {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
           {
             \msg_warning:nnee { tag/debug }{ struct-end-wrong }
              {#1}
              {\exp_last_unbraced:No \use_i:nn { \l__tag_tmpa_tl }}
           }
        }
     }
 }
```

This tracks tag suspend and resume. The tag-suspend message should go before the int is increased. The tag-resume message after the int is decreased.

```
\msg_new:nnn { tag / debug } {tag-suspend}
  {
    \int_if_zero:nTF
      {#1}
      {Tagging~suspended}
      {Tagging~(not)~suspended~(already~inactive)}\\
    level:~#1~==>~\int_eval:n{#1+1}\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
  }
\msg_new:nnn { tag / debug } {tag-resume}
  {
    \int_if_zero:nTF
      {#1}
      {Tagging~resumed}
      {Tagging~(not)~resumed}\\
    level:~\int_eval:n{#1+1}~==>~#1\tl_if_empty:nF{#2}{,~label:~#2}~[\msg_line_context:]
  }
⟨/debug⟩
```

6.6 Benchmarks

It doesn't make much sense to do benchmarks in debug mode or in combination with a log-level as this would slow down the compilation. So we add simple commands that can be activated if l3benchmark has been loaded. TODO: is a warning needed?

```
⟨*package⟩
\cs_new_protected:Npn \__tag_check_benchmark_tic:{}
\cs_new_protected:Npn \__tag_check_benchmark_toc:{}
\cs_new_protected:Npn \tag_check_benchmark_on:
  {
    \cs_if_exist:NT \benchmark_tic:
     {
       \cs_set_eq:NN \__tag_check_benchmark_tic: \benchmark_tic:
       \cs_set_eq:NN \__tag_check_benchmark_toc: \benchmark_toc:
     }
  }
⟨/package⟩
```

Part III

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).

`activate (setup-key)`

And additional setup key which combine the other activate keys `activate/mc`, `activate/tree`, `activate/struct` and additionally adds a document structure.

`\tag_tool:n`
`\tagtool`

`\tag_tool:n {⟨key val⟩}`

The tagging of basic document elements will require a variety of small commands to configure and adapt the tagging. This command will collect them under a command interface. The argument is *one* key-value like string. This is work in progress and both syntax, known arguments and implementation can change!

2 Commands related to mc-chunks

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

```
\tagmcbegin{⟨key-val⟩}
\tagmcend
\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`.

`\tagmcifinTF`

`\tagmcifinTF{⟨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` `\tagstructend` `\tagstructuse`

```
\tagstructbegin{⟨key-val⟩}
\tagstructend
\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 documented in the `tagpdf-struct` module.

4 Debugging

`\ShowTagging`

```
\ShowTagging{⟨key-val⟩}
```

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

`mc-data (show-key)`

```
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 (show-key)`

```
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.

`mc-marks (show-key)`

```
mc-marks = show|use
```

This key helps to debug the page marks. It should only be used at shipout in header or footer.

`struct-stack (show-key)`

```
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`.

`debug/structures (show-key)`

```
debug/structures = ⟨structure number⟩
```

This key is available only if the tagpdf-debug package is loaded and shows all structures starting with the one with the number given by the key.

5 Extension commands

The following commands and code parts are not core commands 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 Tagging of paragraphs

This makes use of the paragraph hooks in 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.

```
para/tagging (setup-key)       para/tagging = true|false
paratagging-show (deprecated)  debug/show=para
paratagging (deprecated)       debug/show=paraOff
```

The `para/tagging` key can be used in `\tagpdfsetup` and enable/disables tagging of paragraphics. `debug/show=para` puts small colored 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.

`\tagpdfsuppressmarks` This command allows to suppress the creation of the marks. It takes an argument which should normally be one of the mc-commands, puts a group around it and suppress the marks creation in this group. This command should be used if the begin and end command are at different boxing levels. E.g.

```
\@hangfrom
 {
  \tagstructbegin{tag=H1}%
  \tagmcbegin    {tag=H1}%
  #2
 }
 {#3\tagpdfsuppressmarks{\tagmcend}\tagstructend}%
```

5.3 Header and footer

Header and footer are automatically tagged as artifact: They are surrounded by an artifact-mc and inside tagging is stopped. If some real content is in the header and footer, tagging must be restarted there explicitly. The behaviour can be changed with the following key. The key accepts the values `true` (the default), `false` which disables the header tagging code. This can be useful if the page style is empty (it then avoids empty mc-chunks) or if the head and foot should be tagged in some special way. The last value, `pagination`, is like `true` but additionally adds an artifact structure with an pagination attribute.

```
page/exclude-header-footer (setup-key) page/exclude-header-footer = true|false|pagination
```

5.4 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 Socket support

```
\tag_socket_use:n     \tag_socket_use:n {⟨socket name⟩}
\tag_socket_use:nnn   \tag_socket_use:nn {⟨socket name⟩} {⟨socket argument⟩}
\UseTaggingSocket     \tag_socket_use:nnn {⟨socket name⟩} {⟨socket argument⟩} {⟨socket argument⟩}
                      \tag_socket_use_expandable:n {⟨socket name⟩}
                      \UseTaggingSocket {⟨socket name⟩}
                      \UseTaggingSocket {⟨socket name⟩} {⟨socket argument⟩}
                      \UseTaggingSocket {⟨socket name⟩} {⟨socket argument⟩} {⟨socket argument⟩}
```

Given that we sometimes have to suspend tagging, it would be fairly inefficient to put different plugs into these sockets whenever that happens. We therefore offer `\UseTaggingSocket` which is like `\UseSocket` except that is expects a socket starting with `tagsupport/` but the socket name is specified without this prefix, i.e.,

`\UseTaggingSocket{foo}` → `\UseSocket{tagsupport/foo}`

.

Beside being slightly shorter, the big advantage is that this way we can change `\UseTaggingSocket` to do nothing by switching a boolean instead of changing the plugs of the tagging support sockets back and forth.

Usually, these sockets have (beside the default plug defined for every socket) one additional plug defined and directly assigned. This plug is used when tagging is active.

There may be more plugs, e.g., tagging with special debugging or special behaviour depending on the class or PDF version etc., but right now it is usually just on or off.

When tagging is suspended they all have the same predefined behaviour: The sockets with zero arguments do nothing. The sockets with one argument gobble their argument. The sockets with two arguments will drop their first argument and pass the second unchanged.

It is possible to use the tagging support sockets with `\UseSocket` directly, but in this case the socket remains active if e.g. `\SuspendTagging` is in force. There may be reasons for doing that but in general we expect to always use `\UseTaggingSocket`.

For special cases like in some `\halign` contexts we need a fully expandable version of the commend. For these cases, `\UseExpandableTaggingSocket` can be used. To allow being expandable, it does not output any debugging information if `\DebugSocketsOn` is in effect and therefore should be avoided whenever possible.

The L3 programming layer versions `\tag_socket_use_expandable:n`, `\tag_socket_use:n`, and `\tag_socket_use:nn`, `\tag_socket_use:nnn` are slightly more efficient than `\UseTaggingSocket` because they do not have to determine how many arguments the socket takes when disabling it.

7 User commands and extensions of document commands

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-user} {2025-06-27} {0.99s}
  {tagpdf - user commands}
⟨/header⟩
```

8 Setup and preamble commands

`\tagpdfsetup`

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

(*End of definition for* `\tagpdfsetup`. *This function is documented on page 39.*)

`\tag_tool:n`
`\tagtool`

This is a first definition of the tool command. Currently it uses key-val, but this should be probably be flattened to speed it up.

```
⟨base⟩\cs_new_protected:Npn\tag_tool:n #1 {}
⟨base⟩\cs_set_eq:NN\tagtool\tag_tool:n
⟨*package⟩
\cs_set_protected:Npn\tag_tool:n #1
  {
    \tag_if_active:T { \keys_set:nn {tag / tool}{#1} }
  }
\cs_set_eq:NN\tagtool\tag_tool:n
⟨/package⟩
```

(*End of definition for* `\tag_tool:n` *and* `\tagtool`*. These functions are documented on page 39.*)

9 Commands for the mc-chunks

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

```
⟨*base⟩
\NewDocumentCommand \tagmcbegin { m }
  {
    \tag_mc_begin:n {#1}
  }


\NewDocumentCommand \tagmcend {  }
  {
    \tag_mc_end:
  }

\NewDocumentCommand \tagmcuse { m }
  {
    \tag_mc_use:n {#1}
  }
⟨/base⟩
```

(*End of definition for* `\tagmcbegin`*,* `\tagmcend`*, and* `\tagmcuse`*. These functions are documented on page 39.*)

`\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.

```
⟨*package⟩
\NewDocumentCommand \tagmcifinTF { m m }
  {
    \tag_mc_if_in:TF { #1 } { #2 }
  }
⟨/package⟩
```

(*End of definition for* `\tagmcifinTF`*. This function is documented on page 39.*)

10 Commands for the structure

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

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

```
⟨*base⟩
\NewDocumentCommand \tagstructbegin { m }
  {
    \tag_struct_begin:n {#1}
  }

\NewDocumentCommand \tagstructend {  }
  {
   \tag_struct_end:
  }
```

```

\NewDocumentCommand \tagstructuse { m }
  {
    \tag_struct_use:n {#1}
  }
⟨/base⟩
```

(*End of definition for* `\tagstructbegin`*,* `\tagstructend`*, and* `\tagstructuse`*. These functions are documented on page 40.*)

11 Socket support

Until we can be sure that the kernel defines the commands we provide them before redefining them: The expandable version will only work correctly after the 2024-11-01 release.

```
⟨*base⟩
\providecommand\tag_socket_use:n[1]{}
\providecommand\tag_socket_use:nn[2]{}
\providecommand\tag_socket_use:nnn[3]{#3}
\providecommand\tag_socket_use_expandable:n[1]{}
\providecommand\socket_use_expandable:nw [1] {
  \use:c { __socket_#1_plug_ \str_use:c { l__socket_#1_plug_str } :w }
}
\providecommand\UseTaggingSocket[1]{}
\providecommand\UseExpandableTaggingSocket[1]{}
⟨/base⟩
```

\tag_socket_use:n
\tag_socket_use:nn
\tag_socket_use:nnn
\UseTaggingSocket
\tag_socket_use_expandable:n
\UseExpandableTaggingSocket

```
⟨*package⟩
\cs_set_protected:Npn \tag_socket_use:n #1
  {
    \bool_if:NT \l__tag_active_socket_bool
      { \socket_use:n {tagsupport/#1} }
  }
\cs_set_protected:Npn \tag_socket_use:nn #1#2
  {
    \bool_if:NT \l__tag_active_socket_bool
       { \socket_use:nn {tagsupport/#1} {#2} }
  }
\cs_set_protected:Npn \tag_socket_use:nnn #1#2#3
  {
    \bool_if:NTF \l__tag_active_socket_bool
      { \socket_use:nnn {tagsupport/#1} {#2} {#3} }
      { #3 }
  }
\cs_set:Npn \tag_socket_use_expandable:n #1
  {
    \bool_if:NT \l__tag_active_socket_bool
      { \socket_use_expandable:n {tagsupport/#1} }
  }
```

```
\cs_set_protected:Npn \UseTaggingSocket #1
  {
    \bool_if:NTF \l__tag_active_socket_bool
      { \socket_use:nw {tagsupport/#1} }
      {
        \int_case:nnF
            { \int_use:c { c__socket_tagsupport/#1_args_int } }
            {
              0 \prg_do_nothing:
              1 \use_none:n
              2 \use_ii:nn
```

We do not expect tagging sockets with more than one or two arguments, so for now we only provide those.

```
            }
            \ERRORusetaggingsocket
      }
  }
\cs_set:Npn \UseExpandableTaggingSocket #1
  {
    \bool_if:NTF \l__tag_active_socket_bool
      { \socket_use_expandable:nw {tagsupport/#1} }
      {
        \int_case:nnF
            { \int_use:c { c__socket_tagsupport/#1_args_int } }
            {
              0 \prg_do_nothing:
              1 \use_none:n
              2 \use_ii:nn
```

We do not expect tagging sockets with more than one or two arguments, so for now we only provide those.

```
            }
            \ERRORusetaggingsocket
      }
  }
⟨/package⟩
```

(*End of definition for* `\tag_socket_use:n` *and others. These functions are documented on page 42.*)

12 Debugging

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

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

  }
```

(*End of definition for* `\ShowTagging`*. This function is documented on page 40.*)

mc-data (show-key) 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 =
      {
        \bool_if:NT \g__tag_mode_lua_bool
          {
            \lua_now:e{ltx.__tag.trace.show_all_mc_data(#1,\__tag_get_mc_abs_cnt:,0)}
          }
      }
    ,mc-data .default:n = 1
  }

```

(*End of definition for* `mc-data (show-key)`*. This function is documented on page 40.*)

mc-current (show-key) 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
          {
            \int_compare:nNnTF
              { -2147483647 }
               =
              {
                \lua_now:e
                  {
                     tex.print
                      (\int_use:N\c_document_cctab,
                       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 of definition for* `mc-current (show-key)`. *This function is documented on page 40.*)

`mc-marks (show-key)` It maps the mc-marks into the sequences and then shows them. This allows to inspect the first and last mc-Mark on a page. It should only be used in the shipout (header/footer).

```
\keys_define:nn { __tag / show }
  {
    mc-marks .choice: ,
    mc-marks / show .code:n =
      {
        \__tag_mc_get_marks:
        \__tag_check_if_mc_in_galley:TF
         {
          \iow_term:n {Marks~from~this~page:~}
         }
         {
           \iow_term:n {Marks~from~a~previous~page:~}
         }
        \seq_show:N \l__tag_mc_firstmarks_seq
        \seq_show:N \l__tag_mc_botmarks_seq
        \__tag_check_if_mc_tmb_missing:T
         {
           \iow_term:n {BDC~missing~on~this~page!}
         }
        \__tag_check_if_mc_tme_missing:T
         {
           \iow_term:n {EMC~missing~on~this~page!}
         }
      },
    mc-marks / use .code:n =
      {
        \__tag_mc_get_marks:
        \__tag_check_if_mc_in_galley:TF
         { Marks~from~this~page:~}
         { Marks~from~a~previous~page:~}
        \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}\quad
        \seq_use:Nn \l__tag_mc_botmarks_seq {,~}\quad
        \__tag_check_if_mc_tmb_missing:T
         {
           BDC~missing~
         }
        \__tag_check_if_mc_tme_missing:T
         {
           EMC~missing
         }
      },
   mc-marks .default:n = show
  }
```

(*End of definition for* `mc-marks (show-key)`. *This function is documented on page 40.*)

`struct-stack (show-key)`

```
\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
  }
⟨/package⟩
```

(*End of definition for* `struct-stack (show-key)`. *This function is documented on page 40.*)

`debug/structures (show-key)` The following key is available only if the tagpdf-debug package is loaded and shows all structures starting with the one with the number given by the key.

```
⟨*debug⟩
\keys_define:nn { __tag / show }
  {
    ,debug/structures .code:n =
      {
        \int_step_inline:nnn{#1}{\c@g__tag_struct_abs_int}
           {
            \msg_term:nneeee
                   { tag/debug } { show-struct }
                   { ##1 }
                   {
                     \prop_map_function:cN
                       {g__tag_struct_debug_##1_prop}
                       \msg_show_item_unbraced:nn
                   }
                   { } { }
            \msg_term:nneeee
                   { tag/debug } { show-kids }
```

```
                    {  ##1 }
                    {
                      \seq_map_function:cN
                        {g__tag_struct_debug_kids_##1_seq}
                        \msg_show_item_unbraced:n
                    }
                    { } { }
           }
       }
     ,debug/structures .default:n = 1
  }
⟨/debug⟩
```

(*End of definition for* `debug/structures (show-key)`*. This function is documented on page 40.*)

13 Commands to extend document commands

The following commands and code parts are not core commands of tagpdf. They 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.

```
⟨*package⟩
```

13.1 Document structure

`\g__tag_root_default_tl`
`activate (setup-key)`
`activate/socket (setup-key)`

```
\tl_new:N\g__tag_root_default_tl
\tl_gset:Nn\g__tag_root_default_tl {Document}

\hook_gput_code:nnn{begindocument}{tagpdf}{\tagstructbegin{tag=\g__tag_root_default_tl}}
\hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}

\keys_define:nn { __tag / setup}
 {
   activate/socket .bool_set:N  = \l__tag_active_socket_bool,
   activate  .code:n =
    {
      \keys_set:nn { __tag / setup }
        { activate/mc,activate/tree,activate/struct,activate/socket }
      \tl_gset:Nn\g__tag_root_default_tl {#1}
    },
   activate .default:n = Document
 }

```

(*End of definition for* `\g__tag_root_default_tl`*,* `activate (setup-key)`*, and* `activate/socket (setup-key)`*. These functions are documented on page 39.*)

13.2 Structure destinations

Since TeXlive 2022 pdftex and luatex offer support for structure destinations and the pdfmanagement has backend support for. We activate them if structures are actually created. Structure destinations are actually PDF 2.0 only but they don't harm in older PDF and can improve html export.

```
\AddToHook{begindocument/before}
  {
    \bool_lazy_and:nnT
      { \g__tag_active_struct_dest_bool }
      { \g__tag_active_struct_bool }
      {
        \tl_set:Nn \l_pdf_current_structure_destination_tl
           { {__tag/struct}{\g__tag_struct_stack_current_tl }}
        \pdf_activate_indexed_structure_destination:
      }
   }
```

13.3 Fake space

\pdffakespace We need a luatex variant for \pdffakespace. This should probably go into the kernel at some time. We also provide a no-op version for dvi mode

```
\bool_if:NT \g__tag_mode_lua_bool
  {
    \NewDocumentCommand\pdffakespace { }
     {
       \__tag_fakespace:
     }
  }
\providecommand\pdffakespace{}
```

(*End of definition for* \pdffakespace. *This function is documented on page 41.*)

13.4 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_flattened_bool
\l__tag_para_show_bool
\g__tag_para_begin_int
\g__tag_para_end_int
\g__tag_para_main_begin_int
\g__tag_para_main_end_int
\g__tag_para_main_struct_tl
\l__tag_para_tag_default_tl
\l__tag_para_tag_tl
\l__tag_para_main_tag_tl
\l__tag_para_attr_class_tl
\l__tag_para_main_attr_class_tl

At first some variables.

```
⟨/package⟩
⟨base⟩\bool_new:N \l__tag_para_flattened_bool
⟨base⟩\bool_new:N \l__tag_para_bool
⟨*package⟩
\int_new:N  \g__tag_para_begin_int
\int_new:N  \g__tag_para_end_int
\int_new:N  \g__tag_para_main_begin_int
\int_new:N  \g__tag_para_main_end_int
```

this will hold the structure number of the current text-unit.

```
\tl_new:N   \g__tag_para_main_struct_tl
\tl_gset:Nn  \g__tag_para_main_struct_tl {1}
\tl_new:N   \l__tag_para_tag_default_tl
\tl_set:Nn  \l__tag_para_tag_default_tl { text }
\tl_new:N   \l__tag_para_tag_tl
\tl_set:Nn  \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
\tl_new:N   \l__tag_para_main_tag_tl
\tl_set:Nn  \l__tag_para_main_tag_tl {text-unit}
```

this is perhaps already defined by the block code

```
\tl_if_exist:NF \l__tag_para_attr_class_tl
 {\tl_new:N \l__tag_para_attr_class_tl }
\tl_new:N \l__tag_para_main_attr_class_tl
```

(*End of definition for* `\l__tag_para_bool` *and others.*)

`\__tag_gincr_para_main_begin_int:`
`\__tag_gincr_para_main_end_int:`
`\__tag_gincr_para_begin_int:`
`\__tag_gincr_para_end_int:`

The global para counter should be set through commands so that `\tag_stop:` can stop them.

```
\cs_new_protected:Npn \__tag_gincr_para_main_begin_int:
 {
   \int_gincr:N \g__tag_para_main_begin_int
 }
\cs_new_protected:Npn \__tag_gincr_para_begin_int:
 {
   \int_gincr:N \g__tag_para_begin_int
 }
\cs_new_protected:Npn \__tag_gincr_para_main_end_int:
 {
   \int_gincr:N \g__tag_para_main_end_int
 }
\cs_new_protected:Npn \__tag_gincr_para_end_int:
 {
   \int_gincr:N \g__tag_para_end_int
 }
```

(*End of definition for* `\__tag_gincr_para_main_begin_int:` *and others.*)

`\__tag_start_para_ints:`
`\__tag_stop_para_ints:`

```
\cs_new_protected:Npn \__tag_start_para_ints:
 {
   \cs_set_protected:Npn \__tag_gincr_para_main_begin_int:
     {
       \int_gincr:N \g__tag_para_main_begin_int
     }
   \cs_set_protected:Npn \__tag_gincr_para_begin_int:
    {
      \int_gincr:N \g__tag_para_begin_int
    }
   \cs_set_protected:Npn \__tag_gincr_para_main_end_int:
    {
      \int_gincr:N \g__tag_para_main_end_int
    }
   \cs_set_protected:Npn \__tag_gincr_para_end_int:
    {
      \int_gincr:N \g__tag_para_end_int
    }
 }
\cs_new_protected:Npn \__tag_stop_para_ints:
 {
   \cs_set_eq:NN \__tag_gincr_para_main_begin_int:\prg_do_nothing:
   \cs_set_eq:NN \__tag_gincr_para_begin_int: \prg_do_nothing:
   \cs_set_eq:NN \__tag_gincr_para_main_end_int: \prg_do_nothing:
   \cs_set_eq:NN \__tag_gincr_para_end_int: \prg_do_nothing:
 }
```

(*End of definition for* `\__tag_start_para_ints:` *and* `\__tag_stop_para_ints:`.)

We want to be able to inspect the current para main structure, so we need a command to store its structure number

`\__tag_para_main_store_struct:`

```
\cs_new:Npn \__tag_para_main_store_struct:
  {
    \tl_gset:Ne \g__tag_para_main_struct_tl {\int_use:N \c@g__tag_struct_abs_int }
  }
```

(*End of definition for* `\__tag_para_main_store_struct:`.)

temporary adaption for the block module:

```
\AddToHook{package/latex-lab-testphase-block/after}
 {
   \tl_if_exist:NT \l_tag_para_attr_class_tl
    {
      \tl_set:Nn \l__tag_para_attr_class_tl { \l_tag_para_attr_class_tl }
    }
}
```

`para/tagging (setup-key)`
`para/tag (setup-key)`
`para/maintag (setup-key)`
`para/tagging (tool-key)`
`para/tag (tool-key)`
`para/maintag (tool-key)`
`para/flattened (tool-key)`
`unittag (deprecated)`
`para-flattened (deprecated)`
`paratagging (deprecated)`
paratagging-show (deprecated)
`paratag (deprecated)`

These keys enable/disable locally paratagging. Paragraphs are typically tagged with two structure: A main structure around the whole paragraph, and inner structures around the various chunks. Debugging can be activated locally with `debug/show=para`, this can affect the typesetting as the small numbers are boxes and they have a (small) height. Debugging can be deactivated with `debug/show=paraOff` The `para/tag` key sets the tag used by the inner structure, `para/maintag` the tag of the outer structure, both can also be changed with `\tag_tool:n`

```
\keys_define:nn { __tag / setup }
  {
    para/tagging      .bool_set:N = \l__tag_para_bool,
    debug/show/para   .code:n = {\bool_set_true:N \l__tag_para_show_bool},
    debug/show/paraOff  .code:n = {\bool_set_false:N \l__tag_para_show_bool},
    para/tag          .tl_set:N   = \l__tag_para_tag_tl,
    para/maintag      .tl_set:N   = \l__tag_para_main_tag_tl,
    para/flattened    .bool_set:N = \l__tag_para_flattened_bool
  }
\keys_define:nn { tag / tool}
  {
    para/tagging   .bool_set:N = \l__tag_para_bool,
    para/tag       .tl_set:N = \l__tag_para_tag_tl,
    para/maintag   .tl_set:N = \l__tag_para_main_tag_tl,
    para/flattened .bool_set:N = \l__tag_para_flattened_bool
  }
```

the deprecated names

```
\keys_define:nn { __tag / setup }
  {
    paratagging      .bool_set:N = \l__tag_para_bool,
    paratagging-show .bool_set:N = \l__tag_para_show_bool,
    paratag          .tl_set:N   = \l__tag_para_tag_tl
  }
\keys_define:nn { tag / tool}
  {
    para    .bool_set:N = \l__tag_para_bool,
    paratag .tl_set:N = \l__tag_para_tag_tl,
    unittag .tl_set:N = \l__tag_para_main_tag_tl,
    para-flattened .bool_set:N = \l__tag_para_flattened_bool
  }
```

(*End of definition for* `para/tagging` *(*`setup-key`*) and others. These functions are documented on page 41.*)

Helper command for debugging:

```
\cs_new_protected:Npn \__tag_check_para_begin_show:nn #1 #2
 %#1 color, #2 prefix
  {
    \bool_if:NT \l__tag_para_show_bool
      {
        \tag_mc_begin:n{artifact}
        \llap{\color_select:n{#1}\tiny#2\int_use:N\g__tag_para_begin_int\ }
        \tag_mc_end:
      }
  }

\cs_new_protected:Npn \__tag_check_para_end_show:nn #1 #2
 %#1  color, #2 prefix
  {
    \bool_if:NT \l__tag_para_show_bool
      {
        \tag_mc_begin:n{artifact}
        \rlap{\color_select:n{#1}\tiny\ #2\int_use:N\g__tag_para_end_int}
        \tag_mc_end:
      }
  }
```

The para/begin and para/end code. We have two variants here: a simpler one, which must be used if the block code is not used (and so probably will disappear at some time) and a more sophisticated one that must be used if the block code is used. It is possible that we will need more variants, so we setup a socket so that the code can be easily switched. This code should move into lttagging, so we add a test for the transition.

```
\str_if_exist:cF { l__socket_tagsupport/para/begin_plug_str }
 {
    \socket_new:nn      {tagsupport/para/begin}{0}
    \socket_new:nn      {tagsupport/para/end}{0}

    \socket_new_plug:nnn{tagsupport/para/begin}{plain}
     {
       \bool_if:NT \l__tag_para_bool
         {
           \bool_if:NF \l__tag_para_flattened_bool
              {
                \__tag_gincr_para_main_begin_int:
                \tag_struct_begin:n
                  {
                    tag=\l__tag_para_main_tag_tl,
                  }
                \__tag_para_main_store_struct:
              }
            \__tag_gincr_para_begin_int:
```

```
          \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
          \__tag_check_para_begin_show:nn {green}{}
          \tag_mc_begin:n {}
       }
   }
 \socket_new_plug:nnn{tagsupport/para/begin}{block}
  {
    \bool_if:NT \l__tag_para_bool
      {
        \legacy_if:nF { @inlabel }
          {
            \__tag_check_typeout_v:n
              {==>~ @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
            \legacy_if:nF { @endpe }
             {
               \bool_if:NF \l__tag_para_flattened_bool
                  {
                    \__tag_gincr_para_main_begin_int:
                    \tag_struct_begin:n
                     {
                      tag=\l__tag_para_main_tag_tl,
                      attribute-class=\l__tag_para_main_attr_class_tl,
                     }
                    \__tag_para_main_store_struct:
                  }
             }
             \__tag_gincr_para_begin_int:
             \__tag_check_typeout_v:n {==>~increment~ P \on@line }
             \tag_struct_begin:n
               {
                 tag=\l__tag_para_tag_tl
                 ,attribute-class=\l__tag_para_attr_class_tl
               }
             \__tag_check_para_begin_show:nn {green}{\PARALABEL}
             \tag_mc_begin:n {}
        }
      }
  }
```

there was no real difference between the original and in the block variant, only a debug message. We therefore define only a plain variant.

```
 \socket_new_plug:nnn{tagsupport/para/end}{plain}
   {
     \bool_if:NT \l__tag_para_bool
       {
         \__tag_gincr_para_end_int:
         \__tag_check_typeout_v:n {==>~increment~ /P \on@line }
         \tag_mc_end:
         \__tag_check_para_end_show:nn {red}{}
         \tag_struct_end:
         \bool_if:NF \l__tag_para_flattened_bool
          {
            \__tag_gincr_para_main_end_int:
            \tag_struct_end:
              }
          }
      }
  }
```

By default we assign the plain plug:

```
\socket_assign_plug:nn { tagsupport/para/begin}{plain}
\socket_assign_plug:nn { tagsupport/para/end}{plain}
```

And use the sockets in the hooks. Once tagging sockets exist, this can be adapted.

```
\AddToHook{para/begin}{ \socket_use:n { tagsupport/para/begin }
  }
\AddToHook{para/end} { \socket_use:n { tagsupport/para/end } }
```

If the block code is loaded we must ensure that it doesn't overwrite the hook again. And we must reassign the para/begin plug. This can go once the block code no longer tries to adapt the hooks.

```
\AddToHook{package/latex-lab-testphase-block/after}
 {
   \RemoveFromHook{para/begin}[tagpdf]
   \RemoveFromHook{para/end}[latex-lab-testphase-block]
   \AddToHook{para/begin}[tagpdf]
    {
      \socket_use:n { tagsupport/para/begin }
    }
  \AddToHook{para/end}[tagpdf]
    {
      \socket_use:n { tagsupport/para/end }
    }
  \socket_assign_plug:nn { tagsupport/para/begin}{block}
 }

```

We check the para count at the end. If tagging is not active it is not a error, but we issue a warning as it perhaps indicates that the testphase code didn't guard everything correctly.

```
\AddToHook{enddocument/info}
  {
    \tag_if_active:F
      {
        \msg_redirect_name:nnn { tag } { para-hook-count-wrong } { warning }
      }
    \int_compare:nNnF {\g__tag_para_main_begin_int}={\g__tag_para_main_end_int}
         {
           \msg_error:nneee
             {tag}
             {para-hook-count-wrong}
             {\int_use:N\g__tag_para_main_begin_int}
             {\int_use:N\g__tag_para_main_end_int}
             {text-unit}
         }
    \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int}
         {
           \msg_error:nneee
             {tag}
             {para-hook-count-wrong}
             {\int_use:N\g__tag_para_begin_int}
             {\int_use:N\g__tag_para_end_int}
             {text}
         }
  }
```

13.5 output routine stuff

We need at least the new-or-1 code. In generic mode we also must insert the code to finish the MC-chunks This part here can go in June 2025

```
\@ifpackageloaded{footmisc}
  {\PackageWarning{tagpdf}{tagpdf~has~been~loaded~too~late!}} %
  {\RequirePackage{latex-lab-testphase-new-or-1}}

\AddToHook{begindocument/before}
 {
   \bool_if:NF \g__tag_mode_lua_bool
     {
        \cs_if_exist:NT \@kernel@before@footins
         {
           \tl_put_right:Nn \@kernel@before@footins
             { \tag_mc_add_missing_to_stream:Nn \footins {footnote} }
           \tl_put_right:Nn \@kernel@tagsupport@@makecol
             {
               \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@
               \tag_mc_add_missing_to_stream:Nn \@outputbox {main}
             }
         }
     }
  }

```

If the new OR is there, we use it

```
\str_if_exist:cT { l__socket_tagsupport/build/column/outputbox_plug_str }
 {
   \NewSocketPlug{tagsupport/build/column/outputbox}{tagpdf}
     {
       \__tag_check_typeout_v:n { ====>~In~\token_to_str:N \@makecol
                                  \c_space_tl\the\c@page }
       \tag_mc_add_missing_to_stream:Nn \@outputbox {main}
     }
   \NewSocketPlug{tagsupport/build/column/footins}{tagpdf}
      { \tag_mc_add_missing_to_stream:Nn \footins {footnote} }

   \bool_if:NF \g__tag_mode_lua_bool
      {
        \AssignSocketPlug{tagsupport/build/column/outputbox}{tagpdf}
        \AssignSocketPlug{tagsupport/build/column/footins}{tagpdf}
      }
  }
⟨/package⟩
```

`\tagpdfparaOn`
`\tagpdfparaOff`

This two command switch para mode on and off. `\tagpdfsetup` could be used too but is longer. An alternative is `\tag_tool:n{para/tagging=false}`

```
⟨base⟩\newcommand\tagpdfparaOn {}
⟨base⟩\newcommand\tagpdfparaOff{}
⟨*package⟩
\renewcommand\tagpdfparaOn {\bool_set_true:N \l__tag_para_bool}
\renewcommand\tagpdfparaOff{\bool_set_false:N \l__tag_para_bool}
```

(*End of definition for* `\tagpdfparaOn` *and* `\tagpdfparaOff`. *These functions are documented on page 41.*)

`\tagpdfsuppressmarks`

This command allows to suppress the creation of the marks. It takes an argument which should normally be one of the mc-commands, puts a group around it and suppress the marks creation in this group. This command should be used if the begin and end command are at different boxing levels. E.g.

```
\@hangfrom
 {
  \tagstructbegin{tag=H1}%
  \tagmcbegin    {tag=H1}%
  #2
 }
 {#3\tagpdfsuppressmarks{\tagmcend}\tagstructend}%
```

```
\NewDocumentCommand\tagpdfsuppressmarks{m}
  {{\use:c{__tag_mc_disable_marks:} #1}}
```

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

13.6 Language support

With the following key the lang variable is set. All structures in the current group will then set this lang variable.

`test/lang (setup-key)`

```
\keys_define:nn { __tag / setup }
  {
    text / lang .tl_set:N = \l__tag_struct_lang_tl
  }
```

(*End of definition for* `test/lang (setup-key)`. *This function is documented on page* ??.)

13.7 Header and footer

Header and footer should normally be tagged as artifacts. The following code requires the new hooks. For now we allow to disable this function, but probably the code should always there at the end. TODO check if Pagination should be changeable.

```
\cs_new_protected:Npn\__tag_hook_kernel_before_head:{}
\cs_new_protected:Npn\__tag_hook_kernel_after_head:{}
\cs_new_protected:Npn\__tag_hook_kernel_before_foot:{}
\cs_new_protected:Npn\__tag_hook_kernel_after_foot:{}
```

This can go once the new OR is active (June 2025)

```
\AddToHook{begindocument}
 {
  \cs_if_exist:NT \@kernel@before@head
   {
     \tl_put_right:Nn \@kernel@before@head {\__tag_hook_kernel_before_head:}
     \tl_put_left:Nn  \@kernel@after@head  {\__tag_hook_kernel_after_head:}
     \tl_put_right:Nn \@kernel@before@foot {\__tag_hook_kernel_before_foot:}
     \tl_put_left:Nn  \@kernel@after@foot  {\__tag_hook_kernel_after_foot:}
   }
 }
```

If the new page sockets exist, we use them.

```
\str_if_exist:cT { l__socket_tagsupport/build/page/footer_plug_str }
 {
   \NewSocketPlug{tagsupport/build/page/header}{tagpdf}
    {
      \__tag_hook_kernel_before_head:
      #2
      \__tag_hook_kernel_after_head:
    }

   \AssignSocketPlug{tagsupport/build/page/header}{tagpdf}
   \NewSocketPlug{tagsupport/build/page/footer}{tagpdf}
   {
     \__tag_hook_kernel_before_foot:
     #2
     \__tag_hook_kernel_after_foot:
   }
  \AssignSocketPlug{tagsupport/build/page/footer}{tagpdf}
 }

\bool_new:N \g__tag_saved_in_mc_bool
\cs_new_protected:Npn \__tag_exclude_headfoot_begin:
 {
    \bool_set_false:N  \l__tag_para_bool
    \bool_if:NTF \g__tag_mode_lua_bool
     {
      \tag_mc_end_push:
     }
     {
       \bool_gset_eq:NN   \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
       \bool_gset_false:N \g__tag_in_mc_bool
     }
    \tag_mc_begin:n {artifact}
    \tag_suspend:n{headfoot}
 }
\cs_new_protected:Npn \__tag_exclude_headfoot_end:
 {
    \tag_resume:n{headfoot}
    \tag_mc_end:
    \bool_if:NTF \g__tag_mode_lua_bool
     {
      \tag_mc_begin_pop:n{}
     }
     {
       \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
     }
 }
```

This version allows to use an Artifact structure

```
\__tag_attr_new_entry:nn {__tag/attr/pagination}{/O/Artifact/Type/Pagination}
\cs_new_protected:Npn \__tag_exclude_struct_headfoot_begin:n #1
 {
    \bool_set_false:N  \l__tag_para_bool
    \bool_if:NTF \g__tag_mode_lua_bool
     {
      \tag_mc_end_push:
     }
     {
       \bool_gset_eq:NN   \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
       \bool_gset_false:N \g__tag_in_mc_bool
     }
    \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1}
    \tag_mc_begin:n {artifact=#1}
    \tag_suspend:n{headfoot}
 }

\cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:
 {
    \tag_resume:n{headfoot}
    \tag_mc_end:
    \tag_struct_end:
    \bool_if:NTF \g__tag_mode_lua_bool
     {
      \tag_mc_begin_pop:n{}
     }
     {
       \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
     }
 }
```

And now the keys

page/exclude-header-footer (setup-key)
exclude-header-footer (deprecated)

```
\keys_define:nn { __tag / setup }
  {
    page/exclude-header-footer .choice:,
    page/exclude-header-footer / true .code:n =
     {
       \cs_set_eq:NN \__tag_hook_kernel_before_head: \__tag_exclude_headfoot_begin:
       \cs_set_eq:NN \__tag_hook_kernel_before_foot: \__tag_exclude_headfoot_begin:
       \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_exclude_headfoot_end:
       \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_exclude_headfoot_end:
     },
    page/exclude-header-footer / pagination .code:n =
     {
       \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_exclude_struct_headfoot_begin:n {p
       \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {p
       \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_exclude_struct_headfoot_end:
       \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_exclude_struct_headfoot_end:
     },
    page/exclude-header-footer / false .code:n =
     {
       \cs_set_eq:NN \__tag_hook_kernel_before_head: \prg_do_nothing:
       \cs_set_eq:NN \__tag_hook_kernel_before_foot: \prg_do_nothing:
       \cs_set_eq:NN \__tag_hook_kernel_after_head:  \prg_do_nothing:
       \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \prg_do_nothing:
     },
   page/exclude-header-footer .default:n = true,
   page/exclude-header-footer .initial:n = true,
```

deprecated name

```
   exclude-header-footer .meta:n = { page/exclude-header-footer = {#1} }
  }
```

(*End of definition for* `page/exclude-header-footer (setup-key)` *and* `exclude-header-footer (deprecated)`. *These functions are documented on page 42.*)

A special, experimental tagged version, which only works with fancyhdr or similar that uses parbox

```
\AtBeginDocument
  {
   \socket_if_exist:nT{tagsupport/parbox/before}
     {
      \NewTaggingSocketPlug{parbox/before}{tag/footer}
        {
         \tag_struct_begin:n{tag=Span}
         \tag_mc_begin:n{}
        }

      \NewTaggingSocketPlug{parbox/after}{tag/footer}
        {
         \tag_mc_end:
         \tag_struct_end:
        }
     }
   }

\cs_new_protected:Npn \__tag_headfoot_tagged_begin:n #1
 {
   \AssignTaggingSocketPlug{parbox/before}{tag/footer}
   \AssignTaggingSocketPlug{parbox/after}{tag/footer}
    \bool_set_false:N  \l__tag_para_bool
    \bool_if:NTF \g__tag_mode_lua_bool
     {
      \tag_mc_end_push:
     }
     {
       \bool_gset_eq:NN   \g__tag_saved_in_mc_bool \g__tag_in_mc_bool
       \bool_gset_false:N \g__tag_in_mc_bool
     }
    \tag_struct_begin:n{tag=Artifact,attribute-class=__tag/attr/#1,parent=\tag_get:n{current_
 }

```

```
\cs_new_protected:Npn \__tag_headfoot_tagged_end:
 {
    \tag_struct_end:
    \bool_if:NTF \g__tag_mode_lua_bool
     {
      \tag_mc_begin_pop:n{}
     }
     {
       \bool_gset_eq:NN \g__tag_in_mc_bool\g__tag_saved_in_mc_bool
     }
 }
\keys_define:nn { __tag / setup }
 {
   page/tag-header-footer .code:n =
    {
      \cs_set:Nn \__tag_hook_kernel_before_head: { \__tag_headfoot_tagged_begin:n {pagination
      \cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_headfoot_tagged_begin:n {pagination
      \cs_set_eq:NN \__tag_hook_kernel_after_head:  \__tag_headfoot_tagged_end:
      \cs_set_eq:NN \__tag_hook_kernel_after_foot:  \__tag_headfoot_tagged_end:
    }
 }
```

13.8 Links

We need to close and reopen mc-chunks around links. We handle URI, GoTo (internal) links, GoToR, Launch and Named links. Links should have an alternative text in the Contents key; this is added for normal links by the generic hyperref driver. With luatex we make use of the lualinksplit package to get OBJR of all annotations into the Link structure, so the hook code should not contain the command to insert the OBJR into the structure.

```
\bool_lazy_and:nnTF
 { \sys_if_engine_luatex_p: }
 {
   \tl_if_empty_p:e
    {
      \lua_now:e
       { if~ luatexbase.in_callback('pre_shipout_filter','linksplit')~
         then~else~tex.print('1')~end
       }
    }
 }
 {
   \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 }
     }

   \hook_gput_code:nnn
     {pdfannot/link/URI/after}
```

```
  {tagpdf}
  {
     \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}
  }

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

\hook_gput_code:nnn
  {pdfannot/link/GoToR/before}
  {tagpdf}
  {
     \tag_mc_end_push:
     \tag_struct_begin:n{tag=Link}
     \tag_mc_begin:n{tag=Link}
  }

\hook_gput_code:nnn
  {pdfannot/link/GoToR/after}
  {tagpdf}
  {
    \tag_mc_end:
    \tag_struct_end:
    \tag_mc_begin_pop:n{}
  }
\hook_gput_code:nnn
  {pdfannot/link/Launch/before}
  {tagpdf}
  {
     \tag_mc_end_push:
     \tag_struct_begin:n{tag=Link}
     \tag_mc_begin:n{tag=Link}
  }

\hook_gput_code:nnn
  {pdfannot/link/Launch/after}
  {tagpdf}
```

```
    {
      \tag_mc_end:
      \tag_struct_end:
      \tag_mc_begin_pop:n{}
    }
  \hook_gput_code:nnn
    {pdfannot/link/Named/before}
    {tagpdf}
    {
       \tag_mc_end_push:
       \tag_struct_begin:n{tag=Link}
       \tag_mc_begin:n{tag=Link}
    }

  \hook_gput_code:nnn
    {pdfannot/link/Named/after}
    {tagpdf}
    {
      \tag_mc_end:
      \tag_struct_end:
      \tag_mc_begin_pop:n{}
    }
}
{
  \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:nne
        { link/URI }
        { StructParent }
        { \tag_struct_parent_int: }
    }

  \hook_gput_code:nnn
    {pdfannot/link/URI/after}
    {tagpdf}
    {
       \tag_struct_insert_annot:ee {\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:nne
         { link/GoTo }
         { StructParent }
         { \tag_struct_parent_int: }
    }

  \hook_gput_code:nnn
    {pdfannot/link/GoTo/after}
    {tagpdf}
    {
      \tag_struct_insert_annot:ee {\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/GoToR/before}
    {tagpdf}
    {
       \tag_mc_end_push:
       \tag_struct_begin:n{tag=Link}
       \tag_mc_begin:n{tag=Link}
       \pdfannot_dict_put:nne
         { link/GoToR }
         { StructParent }
         { \tag_struct_parent_int: }
    }

  \hook_gput_code:nnn
    {pdfannot/link/GoToR/after}
    {tagpdf}
    {
      \tag_struct_insert_annot:ee {\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/Named/before}
    {tagpdf}
    {
       \tag_mc_end_push:
       \tag_struct_begin:n{tag=Link}
       \tag_mc_begin:n{tag=Link}
       \pdfannot_dict_put:nne
         { link/Named }
         { StructParent }
         { \tag_struct_parent_int: }
    }

  \hook_gput_code:nnn
    {pdfannot/link/Named/after}
```

```
    {tagpdf}
    {
      \tag_struct_insert_annot:ee {\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/Launch/before}
    {tagpdf}
    {
       \tag_mc_end_push:
       \tag_struct_begin:n{tag=Link}
       \tag_mc_begin:n{tag=Link}
       \pdfannot_dict_put:nne
         { link/Launch }
         { StructParent }
         { \tag_struct_parent_int: }
    }

  \hook_gput_code:nnn
    {pdfannot/link/Launch/after}
    {tagpdf}
    {
      \tag_struct_insert_annot:ee {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
      \tag_mc_end:
      \tag_struct_end:
      \tag_mc_begin_pop:n{}
    }
 }
```

13.9 Attaching css-files for derivation

Derivation to html (https://pdfa.org/wp-content/uploads/2019/06/Deriving_HTML_-from_PDF.pdf, implemented by, e.g., ngpdf) can be improved by attaching CSS style definitions in associated files with relationship supplement to the Catalog[1].

Such CSS style definitions can be given in two ways:

- In files with the extension `.css`. Such files should contain only CSS style definitions. ngpdf will store these files and include them with an `<link rel=stylesheet href=...>` in the head of the html.
- In files with the extension `.html`. Such files should contain CSS style definitions inside one (or more) `<style>...</style>` html tags. The content of these files are copied by ngpdf directly into the head of the derived html.

By default (if tagging is active) tagpdf embeds now such CSS style definitions. Currently the list of files is rather short and consists of two files (with extension .html and `<style>...</style>` html tags) which are provided by the tagpdf package:

- latex-align-css.html which improves the styling of amsmath alignments tagged with MathML.

[1]Previously they suggested the StructTreeRoot, but this is not compatible with pdf/A-3

- latex-list-css.html which improves the style of list environments.

It is possible to suppress the embedding of these files by setting the `\tagpdfsetup` key `attach-css` to `false`, `attach-css=true` or `attach-css` reverts this again.

For developers, `\tagpdfsetup` some keys to manipulate the list exist: With `css-list={file1,file2}` the list can be overwritten. `css-list=` clears the list (and so suppresses the embedding too). To remove a file from the list, use `css-list-remove=file`, e.g. `css-list-remove=latex-list-css.html`. To add your own file use `css-list-add=my-fancy-align-css.html`. It is also possible to attach a .css-file in this way.

These keys do not affect files added directly with root-supplemental-file or catalog-supplemental-file.

The files in this list are attached at the end of the compilation but you shouldn't rely on a specific order of the embedding in the html.

We want to avoid to embed files twice, so we use a prop.

```
\prop_new:N \g__tag_css_prop
\prop_gset_from_keyval:Nn \g__tag_css_prop
 {
   latex-list-css.html=true,
   latex-align-css.html=true
 }


\bool_new:N \g__tag_css_bool
\bool_gset_true:N \g__tag_css_bool
```

The files for the catalog must be added before the catalog is pushed.

```
\tl_gput_left:Nn \g__kernel_pdfmanagement_end_run_code_tl
 {
  \bool_lazy_and:nnT  { \g__tag_css_bool }{ \tag_if_active_p: }
   {
    \prop_map_inline:Nn \g__tag_css_prop
     {
       \keys_set:nn { __tag / setup }{ catalog-supplemental-file= {#1} }
     }
   }
 }

\keys_define:nn { __tag / setup }
  {
    attach-css .bool_gset:N = \g__tag_css_bool,
    css-list .code:n =
      {
        \tl_if_empty:nTF{#1}
          {\prop_gclear:N \g__tag_css_prop }
          {\prop_gput:Nnn \g__tag_css_prop { #1 }{true}}
      },
    css-list-add .code:n    = { \prop_gput:Nnn \g__tag_css_prop { #1 }{true} },
    css-list-remove .code:n = { \prop_gremove:Nn \g__tag_css_prop { #1 } },
  }
```

</package>

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

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-tree-code} {2025-06-27} {0.99s}
 {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 Check structure

__tag_tree_final_checks:

```
\cs_new_protected:Npn \__tag_tree_final_checks:
 {
   \int_compare:nNnF {\seq_count:N\g__tag_struct_stack_seq}={1}
    {
      \msg_warning:nn {tag}{tree-struct-still-open}
      \int_step_inline:nnn{2}{\seq_count:N\g__tag_struct_stack_seq}
       {\tag_struct_end:}
    }
   \socket_use:n { tag/check/parent-child-end }
   \msg_note:nn {tag}{tree-statistic}
 }
```

(*End of definition for* `\__tag_tree_final_checks:`.)

1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction

The StructTreeRoot and the MarkInfo entry must be added to the catalog. If there is an OpenAction entry we must update it, so that it contains also a structure destination. We do it late so that we can win, but before the pdfmanagement hook.

__tag/struct/1 This is the object for the root object, the StructTreeRoot

```
\pdf_object_new_indexed:nn { __tag/struct }{ 1 }
```

(*End of definition for* __tag/struct/1.)

\g__tag_tree_openaction_struct_tl We need a variable that indicates which structure is wanted in the OpenAction. By default we use 2 (the Document structure).

```
\tl_new:N   \g__tag_tree_openaction_struct_tl
\tl_gset:Nn \g__tag_tree_openaction_struct_tl { 2 }
```

(*End of definition for* \g__tag_tree_openaction_struct_tl.)

viewer/startstructure (setup-key) We also need an option to setup the start structure. So we setup a key which sets the variable to the current structure. This still requires hyperref to do most of the job, this should perhaps be changed.

```
\keys_define:nn { __tag / setup }
{
  viewer/startstructure .code:n =
   {
      \tl_gset:Ne \g__tag_tree_openaction_struct_tl {#1}
   }
 ,viewer/startstructure .default:n =  { \int_use:N \c@g__tag_struct_abs_int }
}
```

(*End of definition for* viewer/startstructure (setup-key). *This function is documented on page* **??**.)

The OpenAction should only be updated if it is there. So we inspect the Catalog-prop:

```
\cs_new_protected:Npn \__tag_tree_update_openaction:
 {
   \prop_get:cnNT
    { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog } }
    {OpenAction}
    \l__tag_tmpa_tl
    {
```

we only do something if the OpenAction is an array (as set by hyperref) in other cases we hope that the author knows what they did.

```
      \tl_if_head_eq_charcode:eNT { \tl_trim_spaces:o { \l__tag_tmpa_tl } } [ %]
        {
          \seq_set_split:Nno\l__tag_tmpa_seq {/} {\l__tag_tmpa_tl}
          \pdfmanagement_add:nne {Catalog} { OpenAction }
            {
              <<
                /S/GoTo \c_space_tl
                /D~\l__tag_tmpa_tl\c_space_tl
                /SD~[\pdf_object_ref_indexed:nn{__tag/struct}{\g__tag_tree_openaction_struct
```

there should be always a /Fit etc in the array but better play safe here ...

```
                      \int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1}
                       { /\seq_item:Nn\l__tag_tmpa_seq{2} }
                       { ] }
              >>
            }
        }
    }
  }
```

```
\hook_gput_code:nnn{shipout/lastpage}{tagpdf}
  {
    \bool_if:NT \g__tag_active_tree_bool
      {
        \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
        \pdfmanagement_add:nne
          { Catalog }
          { StructTreeRoot }
          { \pdf_object_ref_indexed:nn { __tag/struct } { 1 } }
        \__tag_tree_update_openaction:
      }
  }
```

1.3 Writing the IDtree

The ID are currently quite simple: every structure has an ID build from the prefix ID together with the structure number padded with enough zeros to that we get directly an lexical order. We ship them out in bundles At first a seq to hold the references for the kids

\g__tag_tree_id_pad_int

```
\int_new:N\g__tag_tree_id_pad_int
```

(*End of definition for* `\g__tag_tree_id_pad_int`.)

Now we get the needed padding

```
\cs_generate_variant:Nn \tl_count:n {e}
\hook_gput_code:nnn{begindocument}{tagpdf}
 {
   \int_gset:Nn\g__tag_tree_id_pad_int
    {\tl_count:e { \__tag_property_ref_lastpage:nn{tagstruct}{1000}}+1}
 }

```

This is the main code to write the tree it basically splits the existing structure numbers in chunks of length 50 TODO consider is 50 is a good length.

```
\cs_new_protected:Npn \__tag_tree_write_idtree:
  {
    \tl_clear:N \l__tag_tmpa_tl
    \tl_clear:N \l__tag_tmpb_tl
    \int_zero:N \l__tag_tmpa_int
    \int_step_inline:nnn {2} {\c@g__tag_struct_abs_int}
      {
        \int_incr:N\l__tag_tmpa_int
        \tl_put_right:Ne \l__tag_tmpa_tl
```

```
     {
      \__tag_struct_get_id:n{##1}~\pdf_object_ref_indexed:nn {__tag/struct}{##1}~
     }
    \int_compare:nNnF {\l__tag_tmpa_int}<{50} %
     {
      \pdf_object_unnamed_write:ne {dict}
       { /Limits~[\__tag_struct_get_id:n{##1-\l__tag_tmpa_int+1}~\__tag_struct_get_id:
        /Names~[\l__tag_tmpa_tl]
       }
      \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:\c_space_tl}
      \int_zero:N \l__tag_tmpa_int
      \tl_clear:N \l__tag_tmpa_tl
     }
   }
   \tl_if_empty:NF \l__tag_tmpa_tl
   {
    \pdf_object_unnamed_write:ne {dict}
     {
      /Limits~
       [\__tag_struct_get_id:n{\c@g__tag_struct_abs_int-\l__tag_tmpa_int+1}~
        \__tag_struct_get_id:n{\c@g__tag_struct_abs_int}]
      /Names~[\l__tag_tmpa_tl]
     }
    \tl_put_right:Ne\l__tag_tmpb_tl {\pdf_object_ref_last:}
   }
   \pdf_object_unnamed_write:ne {dict}{/Kids~[\l__tag_tmpb_tl]}
   \__tag_prop_gput:cne
     { g__tag_struct_1_prop }
     { IDTree }
     { \pdf_object_ref_last: }
  }
```

1.4 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:cne
    { g__tag_struct_1_prop }
    { ParentTree }
    { \pdf_object_ref:n { __tag/tree/parenttree } }
   \__tag_prop_gput:cne
    { g__tag_struct_1_prop }
    { RoleMap }
    { \pdf_object_ref:n { __tag/tree/rolemap } }
   \__tag_struct_fill_kid_key:n { 1 }
   \prop_gremove:cn { g__tag_struct_1_prop } {S}
   \__tag_struct_get_dict_content:nN { 1 } \l__tag_tmpa_tl
   \pdf_object_write_indexed:nnne
     { __tag/struct } { 1 }
     {dict}
     {
          \l__tag_tmpa_tl
         }
```

Better put S back, see https://github.com/latex3/tagging-project/issues/86

```
      \prop_gput:cnn { g__tag_struct_1_prop } {S}{ /StructTreeRoot }
  }
```

(*End of 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 {2}{1}{\c@g__tag_struct_abs_int}
      {
        \__tag_struct_write_obj:n { ##1 }
      }
  }
```

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

1.5 ParentTree

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

```
\pdf_object_new:n { __tag/tree/parenttree }
```

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

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

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

```
\int_new:N  \c@g__tag_parenttree_obj_int
\hook_gput_code:nnn{begindocument}{tagpdf}
  {
    \int_gset:Nn
      \c@g__tag_parenttree_obj_int
      { \__tag_property_ref_lastpage:nn{abspage}{100}  }
  }
```

(*End of 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 of 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:Ne \g__tag_parenttree_objr_tl
      {
        #1 \c_space_tl #2 ^^J
      }
  }
```

(*End of 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 of 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_parenttree_rerun_msg: {}
\cs_new_protected:Npn \__tag_tree_fill_parenttree:
  {
    \int_step_inline:nnnn{1}{1}{\__tag_property_ref_lastpage:nn{abspage}{-1}} %not quite clea
      { %page ##1
        \prop_clear:N \l__tag_tmpa_prop
        \int_step_inline:nnnn{1}{1}{\__tag_property_ref_lastpage:nn{tagmcabs}{-
1}}
          {
            %mcid####1
            \int_compare:nT
              {\property_ref:enn{mcid-####1}{tagabspage}{-1}=##1} %mcid is on current page
              {% yes
                \prop_get:NnNT
                  \g__tag_mc_parenttree_prop
                  {####1}
                  \l__tag_tmpa_tl
                  {
                   \prop_put:Nee
                    \l__tag_tmpa_prop
                    {\property_ref:enn{mcid-####1}{tagmcid}{-1}}
                    {\l__tag_tmpa_tl}
                  }
              }
          }
        \tl_put_right:Ne\l__tag_parenttree_content_tl
          {
            \int_eval:n {##1-1}\c_space_tl
            [\c_space_tl %]
          }
        \int_step_inline:nnnn %####1
          {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:Ne \l__tag_parenttree_content_tl
                  {
                    \prop_if_exist:cTF  { g__tag_struct_ \l__tag_tmpa_tl _prop  }
                      {
                        \pdf_object_ref_indexed:nn { __tag/struct }{ \l__tag_tmpa_tl }
                      }
                      {
                        null
                      }
                    \c_space_tl
                  }
              }
              {
                \cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg:
                 {
                   \msg_warning:nn { tag } {tree-mcid-index-wrong}
                 }
              }
          }
        \tl_put_right:Nn
          \l__tag_parenttree_content_tl
          {%[
            ]^^J
          }
      }
  }
```

(*End of 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 of 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:
      }
    \__tag_tree_parenttree_rerun_msg:
    \tl_put_right:No \l__tag_parenttree_content_tl { \g__tag_parenttree_objr_tl }
    \pdf_object_write:nne  { __tag/tree/parenttree }{dict}
      {
        /Nums\c_space_tl [\l__tag_parenttree_content_tl]
      }
  }
```

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

1.6 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. Rolemap is also used in PDF 2.0 as a fallback.

```
\pdf_object_new:n { __tag/tree/rolemap }
```

(*End of 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:
 {
   \bool_if:NT \g__tag_role_add_mathml_bool
     {
       \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
        {
          \prop_gput:Nnn \g__tag_role_rolemap_prop {##1}{Span}
        }
     }
   \prop_map_inline:Nn\g__tag_role_rolemap_prop
     {
       \tl_if_eq:nnF {##1}{##2}
        {
          \pdfdict_gput:nne {g__tag_role/RoleMap_dict}
           {##1}
           {\pdf_name_from_unicode_e:n{##2}}
        }
     }
   \pdf_object_write:nne  { __tag/tree/rolemap }{dict}
    {
     \pdfdict_use:n{g__tag_role/RoleMap_dict}
    }
 }
```

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

1.7 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
```

We process the older sec for compatibility with the table code. TODO: check if still needed

```
    \seq_map_inline:Nn \g__tag_attr_class_used_seq
     {
       \prop_gput:Nnn \g__tag_attr_class_used_prop {##1}{}
     }
    \prop_map_inline:Nn \g__tag_attr_class_used_prop
     {
       \prop_get:NnNT \g__tag_attr_entries_prop {##1} \l__tag_tmpb_tl
         {
           \tl_put_right:Ne \l__tag_tmpa_tl
            {
              ##1\c_space_tl
              <<
               \l__tag_tmpb_tl
              >>
             \iow_newline:
            }
         }
     }
    \tl_if_empty:NF
      \l__tag_tmpa_tl
      {
        \pdf_object_new:n { __tag/tree/classmap }
        \pdf_object_write:nne
          { __tag/tree/classmap }
          {dict}
          { \l__tag_tmpa_tl }
        \__tag_prop_gput:cne
          { g__tag_struct_1_prop }
          { ClassMap }
          { \pdf_object_ref:n { __tag/tree/classmap }  }
      }
  }
```

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

1.8 Namespaces

Namespaces are handle in the role module, here is the code to write them out. Namespaces are only relevant for pdf2.0.

__tag/tree/namespaces

```
\pdf_object_new:n { __tag/tree/namespaces }
```

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

__tag_tree_write_namespaces:

```
\cs_new_protected:Npn \__tag_tree_write_namespaces:
  {
   \pdf_version_compare:NnF < {2.0}
    {
      \prop_map_inline:Nn \g__tag_role_NS_prop
        {
          \pdfdict_if_empty:nF {g__tag_role/RoleMapNS_##1_dict}
            {
              \pdf_object_write:nne {__tag/RoleMapNS/##1}{dict}
                {
                  \pdfdict_use:n {g__tag_role/RoleMapNS_##1_dict}
                }
              \pdfdict_gput:nne{g__tag_role/Namespace_##1_dict}
                {RoleMapNS}{\pdf_object_ref:n {__tag/RoleMapNS/##1}}
            }
          \pdf_object_write:nne{tag/NS/##1}{dict}
            {
               \pdfdict_use:n {g__tag_role/Namespace_##1_dict}
            }
        }
      \pdf_object_write:nne {__tag/tree/namespaces}{array}
        {
          \prop_map_tokens:Nn \g__tag_role_NS_prop{\use_ii:nn}
        }
    }
  }
```

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

1.9 Finishing the structure

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

__tag_finish_structure:

```
\hook_new:n {tagpdf/finish/before}
\cs_new_protected:Npn \__tag_finish_structure:
  {
    \bool_if:NT\g__tag_active_tree_bool
      {
        \hook_use:n {tagpdf/finish/before}
        \__tag_tree_final_checks:
        \iow_term:n{Package~tagpdf~Info:~writing~ParentTree}
        \__tag_check_benchmark_tic:
        \__tag_tree_write_parenttree:
        \__tag_check_benchmark_toc:
        \iow_term:n{Package~tagpdf~Info:~writing~IDTree}
        \__tag_check_benchmark_tic:
        \__tag_tree_write_idtree:
        \__tag_check_benchmark_toc:
        \iow_term:n{Package~tagpdf~Info:~writing~RoleMap}
        \__tag_check_benchmark_tic:
        \__tag_tree_write_rolemap:
        \__tag_check_benchmark_toc:
        \iow_term:n{Package~tagpdf~Info:~writing~ClassMap}
        \__tag_check_benchmark_tic:
        \__tag_tree_write_classmap:
        \__tag_check_benchmark_toc:
        \iow_term:n{Package~tagpdf~Info:~writing~NameSpaces}
        \__tag_check_benchmark_tic:
        \__tag_tree_write_namespaces:
        \__tag_check_benchmark_toc:
        \iow_term:n{Package~tagpdf~Info:~writing~StructElems}
        \__tag_check_benchmark_tic:
        \__tag_tree_write_structelements: %this is rather slow!!
        \__tag_check_benchmark_toc:
        \iow_term:n{Package~tagpdf~Info:~writing~Root}
        \__tag_check_benchmark_tic:
        \__tag_tree_write_structtreeroot:
        \__tag_check_benchmark_toc:
      }
  }
⟨/package⟩
```

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

1.10 StructParents entry for Page

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

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

Part V

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:`

```
\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 alt text 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.

`\tag_mc_reset_box:N` ⋆

New: 2023-06-11

```
\tag_mc_reset_box:N ⟨box⟩
```

This resets in lua mode the mc attributes to the one currently in use. It does nothing in generic mode.

`\tag_mc_add_missing_to_stream:Nn`

New: 2024-11-18

`\tag_mc_add_missing_to_stream:Nn` ⟨*box*⟩ `{`⟨*stream name*⟩`}`

This command is only needed in generic mode, in lua mode it gobbles its arguments. In generic mode it adds MC literals to the stream that are missing because of page breaks. The first argument is the box with the stream, the second a string representing the stream. Predeclared are the names `main`, `footnote` and `multicol`. If more streams should be handle the underlying interface must be enabled with `\tag_mc_new_stream:n` The command is only for packages doing deep manipulations of the output routine! Example of use are in the multicol package and in tagpdf itself.

`\tag_mc_new_stream:n`

New: 2024-11-18

`\tag_mc_new_stream:n` `{`⟨*stream name*⟩`}`

This declares the interface needed to handle a new stream with `\tag_mc_add_missing_to_stream:Nn`. Predeclared are the names `main`, `footnote` and `multicol`.

2 Public keys

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

`tag (mc-key)` 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 (mc-key)` 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 (mc-key)` 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.

`alt (mc-key)` 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. The value will be expanded first once. If it is empty, nothing will happen.

`actualtext (mc-key)` 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. The value will be expanded first once. If it is empty, nothing will happen.

`label (mc-key)` 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 (mc-key)` 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 split 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} {2025-06-27} {0.99s}
  {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_abs_int` and `\tl_put_right:Nn\cl@@ckpt{\@elt{g_@@_MCID_abs_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`

```
⟨*base⟩
\newcounter { g__tag_MCID_abs_int }
```

(*End of definition for* `g__tag_MCID_abs_int`.)

`\__tag_get_data_mc_counter:` This command allows `\tag_get:n` to get the current state of the mc counter with the keyword `mc_counter`. By comparing the numbers it can be used to check the number of structure commands in a piece of code.

```
\cs_new:Npn \__tag_get_data_mc_counter:
  {
    \int_use:N \c@g__tag_MCID_abs_int
  }
⟨/base⟩
```

(*End of definition for* `\__tag_get_data_mc_counter:`.)

`\__tag_get_mc_abs_cnt:` A (expandable) function to get the current value of the cnt. TODO: duplicate of the previous one, this should be cleaned up.

```
⟨*shared⟩
\cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }
```

(*End of definition for* `\__tag_get_mc_abs_cnt:`.)

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

```
\bool_new:N \g__tag_in_mc_bool
```

(*End of 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_linked:N \g__tag_mc_parenttree_prop
```

(*End of 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 of 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 of 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 of definition for* `\l__tag_mc_key_stash_bool` *and* `\l__tag_mc_artifact_bool`.)

`\l__tag_mc_lang_tl`

a variable to set a Lang on the mc. This is not conforming to the spec! But it seems to work in acrobat.

```
\tl_new:N \l__tag_mc_lang_tl
```

(*End of definition for* `\l__tag_mc_lang_tl`.)

`\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 of definition for* `\l__tag_mc_key_tag_tl` *and others*.)

3.2 Functions

`\__tag_mc_handle_mc_label:e`

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`. The reference command is based on l3ref.

```
\cs_new:Npn \__tag_mc_handle_mc_label:e #1
  {
    \__tag_property_record:en{tagpdf-#1}{tagabspage,tagmcabs}
  }
```

(*End of definition for* `\__tag_mc_handle_mc_label:e`.)

__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 }
  }
⟨/shared⟩
```

(*End of 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?

```
⟨base⟩\cs_new_protected:Npn \tag_mc_use:n #1 { \__tag_whatsits: }
⟨*shared⟩
\cs_set_protected:Npn \tag_mc_use:n #1 %#1: label name
  {
    \__tag_check_if_active_struct:T
      {
        \tl_set:Ne  \l__tag_tmpa_tl { \property_ref: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:e { \l__tag_tmpa_tl }
                \__tag_mc_set_label_used:n {#1}
              }
              {
                 \msg_warning:nnn {tag}{mc-used-twice}{#1}
              }
          }
      }
  }
⟨/shared⟩
```

(*End of definition for* `\tag_mc_use:n`. *This function is documented on page 79.*)

\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.

```
⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
⟨base⟩\cs_new_protected:Npn \tag_mc_artifact_group_end:{}
⟨*shared⟩
\cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1
 {
  \tag_mc_end_push:
  \tag_mc_begin:n {artifact=#1}
  \group_begin:
  \tag_suspend:n{artifact-group}
 }
```

```

\cs_set_protected:Npn \tag_mc_artifact_group_end:
 {
  \tag_resume:n{artifact-group}
  \group_end:
  \tag_mc_end:
  \tag_mc_begin_pop:n{}
 }
⟨/shared⟩
```

(*End of definition for* `\tag_mc_artifact_group_begin:n` *and* `\tag_mc_artifact_group_end:`. *These functions are documented on page 79.*)

`\tag_mc_reset_box:N` This allows to reset the mc-attributes in box. On base and generic mode it should do nothing.

```
⟨base⟩\cs_new_protected:Npn \tag_mc_reset_box:N #1 {}
```

(*End of definition for* `\tag_mc_reset_box:N`. *This function is documented on page 79.*)

`\tag_mc_end_push:`
`\tag_mc_begin_pop:n`

```
⟨base⟩\cs_new_protected:Npn \tag_mc_end_push: {}
⟨base⟩\cs_new_protected:Npn \tag_mc_begin_pop:n #1 {}
⟨*shared⟩
\cs_set_protected:Npn \tag_mc_end_push:
  {
    \__tag_check_if_active_mc:T
      {
        \__tag_mc_if_in:TF
          {
            \seq_gpush:Ne \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_set_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 of definition for* `\tag_mc_end_push:` *and* `\tag_mc_begin_pop:n`*. These functions are documented on page 79.*)

`\__tag_mc_check_parent_child:n` This checks if an MC can be used in a structure.

```
\cs_new_protected:Npn \__tag_mc_check_parent_child:n #1
% #1 structure number of parent
  {
```

This records if logging is on

```
    \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
      {
        \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
        \msg_note:nnee
         { tag }
         { role-parent-child-check }
         {
           \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
            {??}
            {
              \exp_last_unbraced:No\use_ii:nn
               { \l__tag_get_parent_tmpa_tl }
              :
             \exp_last_unbraced:No\use_i:nn
               { \l__tag_get_parent_tmpa_tl }
            }
         }
         {
           MC~(real~content)
         }
      }
    \__tag_struct_get_role:nnNN
      {#1}
      {rolemap}
      \l__tag_get_parent_tmpa_tl
      \l__tag_get_parent_tmpb_tl
    \__tag_role_check_parent_child:ooooN
      { \l__tag_get_parent_tmpa_tl }
      { \l__tag_get_parent_tmpb_tl }
      { MC } %
      {  } %
      \l__tag_parent_child_check_tl
```

if the return value is 7 we have to check against the parentrole field. TODO ruby and warichu use 7 too, that should be changed!

```
    \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
      {
        \__tag_struct_get_role:nnNN
```

```
          {#1}
          {parentrole}
          \l__tag_get_parent_tmpa_tl
          \l__tag_get_parent_tmpb_tl
        \__tag_role_check_parent_child:ooooN
          { \l__tag_get_parent_tmpa_tl }
          { \l__tag_get_parent_tmpb_tl }
          { MC } %
          {  } %
          \l__tag_parent_child_check_tl
      }
    \__tag_check_forbidden_parent_child:nnee
     {\l__tag_parent_child_check_tl}
     {#1}
     {
       \l__tag_get_parent_tmpb_tl : \l__tag_get_parent_tmpa_tl
     }
     {
       MC~(real content)
     }
  }
\cs_generate_variant:Nn \__tag_mc_check_parent_child:n {o}
```

(*End of definition for* `\__tag_mc_check_parent_child:n`.)

3.3 Keys

This are the keys where the code can be shared between the modes.

`stash (mc-key)`
`__artifact-bool`
`__artifact-type`

the two internal artifact keys are use to define the public `artifact`. For now we add support for the subtypes Header and Footer. Watermark,PageNum, LineNum,Redaction,Bates will be added if some use case emerges. If some use case for /BBox and /Attached emerges, it will be perhaps necessary to adapt the code.

```
\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 / pagination/header .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header }
      },
    __artifact-type / pagination/footer .code:n    =
      {
        \tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }
      },
    __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 of definition for* `stash (mc-key)`*,* `__artifact-bool`*, and* `__artifact-type`*. This function is documented on page 81.*)

```
⟨/shared⟩
```

Part VI
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} {2025-06-27} {0.99s}
 {part of tagpdf - code related to marking chunks - generic mode}
⟨/generic⟩
⟨*debug⟩
\ProvidesExplPackage {tagpdf-debug-generic} {2025-06-27} {0.99s}
 {part of tagpdf - debugging code related to marking chunks - generic mode}
⟨/debug⟩
```

1.1 Variables

```
⟨*generic⟩
```

`\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 of definition for* `\l__tag_mc_ref_abspage_tl`.)

`\l__tag_mc_tmpa_tl` temporary variable

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

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

`\g__tag_mc_marks` a marks register to keep track of the mc's at page breaks and a sequence to keep track of the data for the continuation extra-tmb. We probably will need to track mc-marks in more than one stream, so the seq contains the name of the stream.

```
\newmarks  \g__tag_mc_marks
```

(*End of definition for* `\g__tag_mc_marks`.)

`\g__tag_mc_main_marks_seq`
`\g__tag_mc_footnote_marks_seq`
`\g__tag_mc_multicol_marks_seq`

Each stream has an associated global seq variable holding the bottom marks from the/a previous chunk in the stream. We provide three by default: main, footnote and multicol. TODO: perhaps an interface for more streams will be needed.

```
\seq_new:N \g__tag_mc_main_marks_seq
\seq_new:N \g__tag_mc_footnote_marks_seq
\seq_new:N \g__tag_mc_multicol_marks_seq
```

(*End of definition for* `\g__tag_mc_main_marks_seq`*,* `\g__tag_mc_footnote_marks_seq`*, and* `\g__tag_mc_multicol_marks_seq`.)

\tag_mc_new_stream:n

```
\cs_new_protected:Npn  \tag_mc_new_stream:n #1
  {
    \seq_new:c { g__tag_mc_multicol_#1_seq }
  }
```

(*End of definition for* `\tag_mc_new_stream:n`*. This function is documented on page 80.*)

\l__tag_mc_firstmarks_seq
\l__tag_mc_botmarks_seq

The marks content contains a number of data which we will have to access and compare, so we will store it locally in two sequences. topmarks is unusable in LaTeX so we ignore it.

```
\seq_new:N  \l__tag_mc_firstmarks_seq
\seq_new:N  \l__tag_mc_botmarks_seq
```

(*End of definition for* `\l__tag_mc_firstmarks_seq` *and* `\l__tag_mc_botmarks_seq`*.*)

1.2 Functions

__tag_mc_begin_marks:nn
__tag_mc_artifact_begin_marks:n
__tag_mc_end_marks:

Generic mode need to set marks for the page break and split stream handling. We always set two marks to be able to detect the case when no mark is on a page/galley. MC-begin commands will set (b,-,data) and (b,+,data), MC-end commands will set (e,-,data) and (e,+,data).

```
\cs_new_protected:Npn \__tag_mc_begin_marks:nn #1 #2 %#1 tag, #2 label
  {
    \tex_marks:D \g__tag_mc_marks
      {
        b-, %first of begin pair
        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
        \g__tag_struct_stack_current_tl,  %structure num
        #1, %tag
        \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
        #2, %label
      }
    \tex_marks:D \g__tag_mc_marks
      {
        b+, % second of begin pair
        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
        \g__tag_struct_stack_current_tl,  %structure num
        #1, %tag
        \bool_if:NT \l__tag_mc_key_stash_bool{stash}, % stash info
        #2, %label
      }
  }
\cs_generate_variant:Nn \__tag_mc_begin_marks:nn {oo}
\cs_new_protected:Npn \__tag_mc_artifact_begin_marks:n #1 %#1 type
  {
    \tex_marks:D \g__tag_mc_marks
      {
        b-, %first of begin pair
        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
        -1, %structure num
        #1 %type
      }
    \tex_marks:D \g__tag_mc_marks
      {
        b+, %first of begin pair
        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
        -1,  %structure num
        #1  %Type
      }
  }

\cs_new_protected:Npn \__tag_mc_end_marks:
  {
    \tex_marks:D \g__tag_mc_marks
      {
        e-, %first of end pair
        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
        \g__tag_struct_stack_current_tl,  %structure num
      }
    \tex_marks:D \g__tag_mc_marks
      {
        e+, %second of end pair
        \int_use:N\c@g__tag_MCID_abs_int, %mc-num
        \g__tag_struct_stack_current_tl,  %structure num
      }
  }
```

(*End of definition for* `\__tag_mc_begin_marks:nn`*,* `\__tag_mc_artifact_begin_marks:n`*, and* `\__tag_mc_end_marks:`*.*)

`\__tag_mc_disable_marks:` This disables the marks. They can't be reenabled, so it should only be used in groups.

```
\cs_new_protected:Npn \__tag_mc_disable_marks:
 {
   \cs_set_eq:NN \__tag_mc_begin_marks:nn \use_none:nn
   \cs_set_eq:NN \__tag_mc_artifact_begin_marks:n \use_none:n
   \cs_set_eq:NN \__tag_mc_end_marks: \prg_do_nothing:
 }
```

(*End of definition for* `\__tag_mc_disable_marks:`*.*)

`\__tag_mc_get_marks:` This stores the current content of the marks in the sequences. It naturally should only be used in places where it makes sense.

```
\cs_new_protected:Npn \__tag_mc_get_marks:
 {
   \exp_args:NNe
   \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
     { \tex_firstmarks:D \g__tag_mc_marks }
   \exp_args:NNe
   \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
     { \tex_botmarks:D   \g__tag_mc_marks }
 }
```

(*End of definition for* `\__tag_mc_get_marks:`*.*)

`\__tag_mc_store:nnn` This inserts the mc-chunk ⟨*mc-num*⟩ into the structure struct-num after the ⟨*mc-prev*⟩. The structure must already exist. The additional mcid dictionary is stored in a property.

The item is retrieved when the kid entry is built. We test if there is already an addition and append if needed.

```
\cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-
num
  {
    %\prop_show:N \g__tag_struct_cont_mc_prop
    \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
      {
        \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_
      }
      {
        \prop_gput:Nne \g__tag_struct_cont_mc_prop {#1}{ \__tag_struct_mcid_dict:n {#2}}
      }
    \prop_gput:Nee \g__tag_mc_parenttree_prop
      {#2}
      {#3}
  }
\cs_generate_variant:Nn \__tag_mc_store:nnn {eee}
```

(*End of definition for* `\__tag_mc_store:nnn`.)

`\__tag_mc_insert_extra_tmb:n`
`\__tag_mc_insert_extra_tme:n`

These two functions should be used in the output routine at the place where a mc-literal could be missing due to a page break or some other split. They check (with the help of the marks) if a extra-tmb or extra-tme is needed. The tmb command stores also the mc into the structure, the tme has to store the data for a following extra-tmb. The argument takes a stream name like main or footnote to allow different handling there. The content of the marks must be stored before (with `\@@_mc_get_marks:` or manually) into `\l_@@_mc_firstmarks_seq` and `\l_@@_mc_botmarks_seq` so that the tests can use them.

```
\cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
  {
     \__tag_check_typeout_v:n {=>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
     \__tag_check_typeout_v:n {=>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
     \__tag_check_if_mc_tmb_missing:TF
       {
         \__tag_check_typeout_v:n {=>~ TMB~ ~ missing~ ---~ inserted}
         %test if artifact
         \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {-
1}
           {
              \tl_set:Ne \l__tag_tmpa_tl { \seq_item:cn { g__tag_mc_#1_marks_seq } {4} }
              \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
           }
           {
              \exp_args:Ne
              \__tag_mc_bdc_mcid:n
                {
                  \seq_item:cn { g__tag_mc_#1_marks_seq } {4}
                }
              \str_if_eq:eeTF
                {
                  \seq_item:cn { g__tag_mc_#1_marks_seq } {5}
                }
                {}
                {
                  %store
                  \__tag_mc_store:eee
                    {
                      \seq_item:cn { g__tag_mc_#1_marks_seq } {2}
                    }
                    { \int_eval:n{\c@g__tag_MCID_abs_int} }
                    {
                      \seq_item:cn { g__tag_mc_#1_marks_seq } {3}
                    }
                }
                {
                   %stashed -> warning!!
                }
           }
       }
       {
         \__tag_check_typeout_v:n {=>~ TMB~ not~ missing}
       }
  }

\cs_new_protected:Npn \__tag_mc_insert_extra_tme:n #1 % #1 stream, eg. main or footnote
 {
   \__tag_check_if_mc_tme_missing:TF
     {
       \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ ---~ inserted}
       \__tag_mc_emc:
       \seq_gset_eq:cN
         {  g__tag_mc_#1_marks_seq }
         \l__tag_mc_botmarks_seq
     }
     {
       \__tag_check_typeout_v:n {=>~ TME~ not~ missing}
     }
 }
```

(*End of definition for* `\__tag_mc_insert_extra_tmb:n` *and* `\__tag_mc_insert_extra_tme:n`.)

1.3 Looking at MC marks in boxes

`\__tag_add_missing_mcs:Nn`

Assumptions:

- test for tagging active outside;
- mark retrieval also outside.

This takes a box register as its first argument (or the register number in a count register, as used by multicol). It adds an extra tmb at the top of the box if necessary and similarly an extra tme at the end. This is done by adding hboxes in a way that the positioning and the baseline of the given box is not altered. The result is written back to the box.

The second argument is the stream this box belongs to und is currently either `main` for the main galley, `footnote` for footnote note text, or `multicol` for boxes produced for columns in that environment. Other streams may follow over time.

```
\cs_new_protected:Npn\__tag_add_missing_mcs:Nn #1 #2 {
  \vbadness \@M
  \vfuzz    \c_max_dim
  \vbox_set_to_ht:Nnn #1 { \box_ht:N #1 } {
    \hbox_set:Nn \l__tag_tmpa_box { \__tag_mc_insert_extra_tmb:n {#2} }
    \hbox_set:Nn \l__tag_tmpb_box { \__tag_mc_insert_extra_tme:n {#2} }
    \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
         {
           \seq_log:c { g__tag_mc_#2_marks_seq}
         }
```

The box placed on the top gets zero size and thus will not affect the box dimensions of the box we are modifying.

```
    \box_set_ht:Nn \l__tag_tmpa_box \c_zero_dim
    \box_set_dp:Nn \l__tag_tmpa_box \c_zero_dim
```

The box added at the bottom will get the depth of the original box. This way we can arrange that from the outside everything looks as before.

```
    \box_set_ht:Nn \l__tag_tmpb_box \c_zero_dim
    \box_set_dp:Nn \l__tag_tmpb_box { \box_dp:N #1 }
```

We need to set `\boxmaxdepth` in case the original box has an unusually large depth, otherwise that depth is not preserved when we string things together.

```
    \boxmaxdepth \@maxdepth
    \box_use_drop:N        \l__tag_tmpa_box
    \vbox_unpack_drop:N     #1
```

Back up by the depth of the box as we add that later again.

```
    \tex_kern:D -\box_dp:N \l__tag_tmpb_box
```

And we don't want any glue added when we add the box.

```
    \nointerlineskip
    \box_use_drop:N \l__tag_tmpb_box
  }
}
```

(*End of definition for* `\__tag_add_missing_mcs:Nn`.)

`\tag_mc_add_missing_to_stream:Nn`
`\__tag_add_missing_mcs_to_stream:Nn`

This is the main command to add mc to the stream. It is therefore guarded by the mc-boolean.

If we aren't in the main stream then processing is a bit more complicated because to get at the marks in the box we need to artificially split it and then look at the split marks.

First argument is the box to update and the second is the "stream". In lua mode the command is a no-op.

```
\cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2
  {
    \__tag_check_if_active_mc:T {
```

First set up a temp box for trial splitting.

```
    \vbadness\maxdimen
    \box_set_eq:NN \l__tag_tmpa_box #1
```

Split the box to the largest size available. This should give us all content (but to be sure that there is no issue we could test out test box is empty now (not done).

```
    \vbox_set_split_to_ht:NNn \l__tag_tmpa_box \l__tag_tmpa_box \c_max_dim
```

As a side effect of this split we should now have the first and bottom split marks set up. We use this to set up `\l__tag_mc_firstmarks_seq`

```
    \exp_args:NNe
    \seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq
        { \tex_splitfirstmarks:D \g__tag_mc_marks }
```

Some debugging info:

```
%    \iow_term:n { First~ mark~ from~ this~ box: }
%    \seq_log:N \l__tag_mc_firstmarks_seq
```

If this mark was empty then clearly the bottom mark will too be empty. Thus in this case we make use of the saved bot mark from the previous chunk. Note that if this is the first chunk in the stream the global seq would contain a random value, but then we can't end in this branch because the basis assumption is that streams are properly marked up so the first chunk would always have a mark at the beginning!

```
    \seq_if_empty:NTF \l__tag_mc_firstmarks_seq
        {
          \__tag_check_typeout_v:n
            {
              No~ marks~ so~ use~ saved~ bot~ mark:~
              \seq_use:cn {g__tag_mc_#2_marks_seq} {,~} \iow_newline:
            }
         \seq_set_eq:Nc \l__tag_mc_firstmarks_seq {g__tag_mc_#2_marks_seq}
```

We also update the bot mark to the same value so that we can later apply `\__tag_add_-missing_mcs:Nn` with the data structures in place (see assumptions made there).

```
          \seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
        }
```

If there was a first mark then there is also a bot mark (and it can't be the same as our marks always come in pairs). So if that branch is chosen we update `\l__tag_mc_-botmarks_seq` from the bot mark.

```
        {
          \__tag_check_typeout_v:n
            {
              Pick~ up~ new~ bot~ mark!
            }
          \exp_args:NNe
          \seq_set_from_clist:Nn \l__tag_mc_botmarks_seq
              { \tex_splitbotmarks:D   \g__tag_mc_marks }
        }
```

Finally we call `\__tag_add_missing_mcs:Nn` to add any missing tmb/tme as needed,

```
    \__tag_add_missing_mcs:Nn #1 {#2}
%%
    \seq_gset_eq:cN  {g__tag_mc_#2_marks_seq} \l__tag_mc_botmarks_seq
%%
  }
}
\cs_set_eq:NN \tag_mc_add_missing_to_stream:Nn \__tag_add_missing_mcs_to_stream:Nn
```

(*End of definition for* `\tag_mc_add_missing_to_stream:Nn` *and* `\__tag_add_missing_mcs_to_stream:Nn`. *This function is documented on page 80.*)

`\__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.

One exception are header and footer (perhaps they are more, but for now it doesn't seem so, so there are no dedicated code to handle this situation): When they are built and added to the page we could be both inside or outside a mc-chunk. But header and footer should ignore this and not push/pop or warn about nested mc. It is therefore important there to set and reset the boolean manually. See the tagpddocu-patches.sty for an example.

```
\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 of definition for* `\__tag_mc_if_in:TF` *and* `\tag_mc_if_in:TF`*. This function is documented on page 79.*)

`\__tag_mc_bmc:n`
`\__tag_mc_emc:`
`\__tag_mc_bdc:nn`

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. change 2023-08-18: we are delaying the writing to the shipout.

```
% #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_set_eq:NN \__tag_mc_bdc_shipout:ee \pdf_bdc_shipout:ee
```

(*End of 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:oo`

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. Starting with texlive 2023 this is much simpler and faster as we can use delay the numbering to the shipout. We also define a wrapper around the low-level command as luamode will need something different.

```
\hook_gput_code:nnn {shipout/before}{tagpdf}{ \flag_clear:n { __tag/mcid } }
\cs_set_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
  {
    \int_gincr:N \c@g__tag_MCID_abs_int
    \__tag_property_record:eo
     {
       mcid-\int_use:N \c@g__tag_MCID_abs_int
     }
     { \c__tag_property_mc_clist }
    \__tag_mc_bdc_shipout:ee
      {#1}
      {
        /MCID~\flag_height:n { __tag/mcid }
        \flag_raise:n { __tag/mcid }~  #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 {oo}
```

(*End of 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:e`

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:Nee \g__tag_mc_parenttree_prop
     {#1}
     { \g__tag_struct_stack_current_tl }
  }
\cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }
```

(*End of 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
  {
    \int_gincr:N \c@g__tag_MCID_abs_int
    \tl_if_empty:NTF #1
      { \__tag_mc_bmc_artifact: }
      { \exp_args:No\__tag_mc_bmc_artifact:n {#1} }
  }
```

(*End of 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 }
⟨/generic⟩
```

(*End of 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.

```
⟨base⟩\cs_new_protected:Npn \tag_mc_begin:n #1 { \__tag_whatsits: \int_gincr:N \c@g__tag_MCID_
⟨base⟩\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
⟨*generic | debug⟩
⟨*generic⟩
\cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
  {
    \__tag_check_if_active_mc:T
      {
⟨/generic⟩
⟨*debug⟩
\cs_set_protected:Npn \tag_mc_begin:n #1 %#1 keyval
  {
    \__tag_check_if_active_mc:TF
      {
        \__tag_debug_mc_begin_insert:n { #1 }
⟨/debug⟩
        \group_begin: %hm
        \__tag_check_mc_if_nested:
        \bool_gset_true:N \g__tag_in_mc_bool
```

set default MC tags to structure:

```
        \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
        \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
        \tl_if_empty:NTF\l__tag_mc_lang_tl
          {
            \keys_set:nn { __tag / mc }{ #1 }
          }
          {
            \keys_set:nn { __tag / mc }{ lang=\l__tag_mc_lang_tl, #1 }
          }
        \bool_if:NTF \l__tag_mc_artifact_bool
          { %handle artifact
            \__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
            \exp_args:No
            \__tag_mc_artifact_begin_marks: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:oo
              { \l__tag_mc_key_tag_tl }
              { \l__tag_mc_key_properties_tl }
            \__tag_mc_begin_marks:oo{\l__tag_mc_key_tag_tl}{\l__tag_mc_key_label_tl}
            \tl_if_empty:NF {\l__tag_mc_key_label_tl}
              {
                \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
              }
```

check if the MC can be used here. This is guarded by the stash boolean.

```
              \bool_if:NF \l__tag_mc_key_stash_bool
                {
                  \socket_use:nn{tag/check/parent-child}
                    {
                     \__tag_mc_check_parent_child:o
                      { \g__tag_struct_stack_current_tl }
                    }
                  \__tag_mc_handle_stash:e { \int_use:N \c@g__tag_MCID_abs_int }

                }
            }
        \group_end:
      }
⟨*debug⟩
      {
        \__tag_debug_mc_begin_ignore:n { #1 }
      }
⟨/debug⟩
  }
⟨*generic⟩
\cs_set_protected:Nn \tag_mc_end:
  {
    \__tag_check_if_active_mc:T
      {
⟨/generic⟩
⟨*debug⟩
\cs_set_protected:Nn \tag_mc_end:
  {
    \__tag_check_if_active_mc:TF
      {
        \__tag_debug_mc_end_insert:
⟨/debug⟩
        \__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:
        \__tag_mc_end_marks:
      }
⟨*debug⟩
      {
        \__tag_debug_mc_end_ignore:
      }
⟨/debug⟩
  }
⟨/generic | debug⟩
```

(*End of definition for* `\tag_mc_begin:n` *and* `\tag_mc_end:`*. These functions are documented on page 79.*)

1.4 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 (mc-key)`
`raw (mc-key)`
`alt (mc-key)`
`actualtext (mc-key)`
`label (mc-key)`
`artifact (mc-key)`

```
⟨*generic⟩
\keys_define:nn { __tag / mc }
  {
    tag .code:n = % the name (H,P,Span) etc
      {
        \tl_set:Ne   \l__tag_mc_key_tag_tl { #1 }
        \tl_gset:Ne  \g__tag_mc_key_tag_tl { #1 }
      },
    raw  .code:n =
      {
        \tl_put_right:Ne \l__tag_mc_key_properties_tl { #1 }
      },
    alt .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>~ }
      },
    alttext .meta:n = {alt=#1},
```

lang is not according to the spec, but it works in acrobat .... We assume that this are simple strings that do not need escaping.

```
    lang .code:n      = % Lang property
      {
        \tl_put_right:Ne \l__tag_mc_key_properties_tl { /Lang~(#1) }
      },
    actualtext .code:n      = % ActualText property
      {
        \tl_if_empty:oF{#1}
         {
           \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:Nne
          \keys_set:nn
            { __tag / mc }
            { __artifact-bool, __artifact-type=#1 }
      },
    artifact .default:n    = {notype}
  }
⟨/generic⟩
```

(*End of definition for* `tag (mc-key)` *and others. These functions are documented on page 80.*)

Part VII

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

The code is split 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} {2025-06-27} {0.99s}
  {tagpdf - mc code only for the luamode }
⟨/luamode⟩
⟨*debug⟩
\ProvidesExplPackage {tagpdf-debug-lua} {2025-06-27} {0.99s}
 {part of tagpdf - debugging code related to marking chunks - lua mode}
⟨/debug⟩
```

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")~
              if~luatexbase.declare_callback_rule~then~
                luatexbase.declare_callback_rule("pre_shipout_filter", "luaotfload.dvi", "aft
              end~
            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_add_missing_mcs_to_stream:Nn

This command is used in the output routine by the ptagging code. It should do nothing in luamode.

```
\cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2 {}
\cs_set_eq:NN \tag_mc_add_missing_to_stream:Nn \__tag_add_missing_mcs_to_stream:Nn
```

(*End of definition for* __tag_add_missing_mcs_to_stream:Nn.)

\tag_mc_new_stream:n

```
\cs_new_protected:Npn  \tag_mc_new_stream:n #1 {}
```

(*End of definition for* \tag_mc_new_stream:n. *This function is documented on page 80.*)

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

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(\int_use:N \c_document_cctab,tex.getattribute(luatexbase.attributes.g__t
         }
      }
      { \prg_return_false:  }
      { \prg_return_true: }
  }

\prg_new_eq_conditional:NNn \tag_mc_if_in: \__tag_mc_if_in: {p,T,F,TF}
```

(*End of definition for* __tag_mc_if_in:TF *and* \tag_mc_if_in:TF. *This function is documented on page 79.*)

__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 globally to the related number.

```
\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:Ne\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 of 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 of 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:e`

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

```
⟨/luamode⟩
⟨*luamode | debug⟩
```

```
⟨luamode⟩\cs_new_protected:Npn \__tag_mc_handle_stash:n #1 %1 mcidnum
⟨debug⟩\cs_set_protected:Npn \__tag_mc_handle_stash:n #1 %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}%
      }
⟨debug⟩    \seq_gput_right:cn % Don't fill a lua table due to the command in the item,
⟨debug⟩                       % so use the kernel command
⟨debug⟩      { g__tag_struct_debug_kids_\g__tag_struct_stack_current_tl _seq }
⟨debug⟩      {
⟨debug⟩        MC~#1%
⟨debug⟩      }
    \lua_now:e
      {
        ltx.__tag.func.store_struct_mcabs
          (
            \g__tag_struct_stack_current_tl,#1
          )
      }
  }
⟨/luamode | debug⟩
⟨*luamode⟩
\cs_generate_variant:Nn \__tag_mc_handle_stash:n { e }
```

(*End of 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_set_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
```

set the default tag to the structure:

```
        \tl_set_eq:NN \l__tag_mc_key_tag_tl \g__tag_struct_tag_tl
        \tl_gset_eq:NN\g__tag_mc_key_tag_tl \g__tag_struct_tag_tl
        \lua_now:e
          {
            ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"tag","\g__tag_struct_tag_tl'
          }
```

2025-05-23 allow lang on the MC (not really spec conform, but does work in acrobat).

```
        \tl_if_empty:NTF\l__tag_mc_lang_tl
          {
            \keys_set:nn { __tag / mc }{ label={}, #1 }
          }
          {
            \keys_set:nn { __tag / mc }{ label={},lang=\l__tag_mc_lang_tl, #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}
              {
                 \__tag_mc_handle_mc_label:e { \l__tag_mc_key_label_tl }
              }
           % if not stashed record the absolute number
            \bool_if:NF \l__tag_mc_key_stash_bool
              {
                \socket_use:nn{tag/check/parent-child}
                  {
                    \__tag_mc_check_parent_child:o
                     { \g__tag_struct_stack_current_tl }
                  }
                \__tag_mc_handle_stash:e { \__tag_get_mc_abs_cnt: }
              }
          }
        \group_end:
     }
  }
```

(*End of definition for* `\tag_mc_begin:n`. *This function is documented on page 79.*)

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

```
\cs_set_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 of definition for* `\tag_mc_end:`. *This function is documented on page 79.*)

`\tag_mc_reset_box:N` This allows to reset the mc-attributes in box. On base and generic mode it should do nothing.

```
\cs_set_protected:Npn \tag_mc_reset_box:N #1
  {
    \lua_now:e
     {
       local~type=tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr)
```

```
      local~mc=tex.getattribute(luatexbase.attributes.g__tag_mc_cnt_attr)
      ltx.__tag.func.update_mc_attributes(tex.getbox(\int_use:N #1),mc,type)
    }
 }
```

(*End of definition for* `\tag_mc_reset_box:N`. *This function is documented on page 79.*)

`\__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 of definition for* `\__tag_get_data_mc_tag:`.)

1.2 Key definitions

`tag (mc-key)`
`raw (mc-key)`
`alt (mc-key)`
`lang (mc-key=`
`actualtext (mc-key)`
`label (mc-key)`
`artifact (mc-key)`

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

```
\keys_define:nn { __tag / mc }
  {
    tag .code:n = %
      {
        \tl_set:Ne   \l__tag_mc_key_tag_tl { #1 }
        \tl_gset:Ne  \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:Ne \l__tag_mc_key_properties_tl { #1 }
        \lua_now:e
          {
            ltx.__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:,"raw","#1")
          }
      },
    alt .code:n      = % Alt property
      {
        \tl_if_empty:oF{#1}
          {
            \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>"
                  )
              }
           }
      },
    lang .code:n      = % Lang property
          {
            \tl_if_empty:oF{#1}
              {
                \tl_put_right:Ne \l__tag_mc_key_properties_tl { /Lang~(#1) }
                \lua_now:e
                  {
                    ltx.__tag.func.store_mc_data
                      (
                        \__tag_get_mc_abs_cnt:,"lang","/Lang~(#1)"
                      )
                  }
               }
          },
    alttext .meta:n = {alt=#1},
    actualtext .code:n      = % Alt property
      {
        \tl_if_empty:oF{#1}
          {
            \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:Nne
          \keys_set:nn
            { __tag / mc}
            { __artifact-bool, __artifact-type=#1, tag=Artifact }
        \exp_args:Nne
          \keys_set:nn
            { __tag / mc }
            { __artifact-store=\l__tag_mc_artifact_type_tl }
      },
    artifact .default:n    = { notype }
  }

⟨/luamode⟩
```

(*End of definition for* `tag (mc-key)` *and others. These functions are documented on page 80.*)

Part VIII
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_end:n`

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

These commands start and end a new structure. They don't start a group. They set all their values globally. `\tag_struct_end:n` does nothing special normally (apart from swallowing its argument, but if `tagpdf-debug` is loaded, it will check if the `{`⟨*tag*⟩`}` (after expansion) is identical to the current structure on the stack. The tag is not role mapped!

`\tag_struct_use:n`
`\tag_struct_use_num:n`

```
\tag_struct_use:n {⟨label⟩}
\tag_struct_use_num:n {⟨structure number⟩}
```

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.

`\tag_struct_object_ref:n`
`\tag_struct_object_ref:e`

```
\tag_struct_object_ref:n {⟨structure number⟩}
```

This is a small wrapper around `\pdf_object_ref:n` to retrieve the object reference of the structure with the number ⟨*struct number*⟩. This number can be retrieved and stored for the current structure for example with `\tag_get:n{`⟨$struct_n um$⟩`}`. Be aware that it can only be used if the structure has already been created and that it doesn't check if the object actually exists!

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 annotation. 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.

`\tag_struct_gput:nnn` `\tag_struct_gput:nnn {⟨structure number⟩} {⟨keyword⟩} {⟨value⟩}`

This is a command that allows to update the data of a structure. This often can't done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the only keyword is `ref` which updates the Ref key (an array)

`\tag_struct_gput_ref:nnn` `\tag_struct_gput_ref:nnn {⟨structure number⟩} {⟨keyword⟩} {⟨value⟩}`

This is an user interface to add a Ref key to an existing structure. The target structure doesn't have to exist yet but can be addressed by label, destname or even num. ⟨*keyword*⟩ is currently either `label`, `dest` or `num`. The value is then either a label name, the name of a destination or a structure number.

2 Public keys

2.1 Keys for the structure commands

`tag` (*struct key*) This is required. The value of the key is normally one of the standard types listed in the main tagpdf documentation. 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` (*struct key*) 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` (*struct key*) This key sets a label by which one can refer to the structure. It is e.g. used by `\tag_struct_use:n` (where a real label is actually not needed as you can only use structures already defined), and by the `ref` key (which can refer to future structures). Internally the label name will start with `tagpdfstruct-` and it stores the two attributes `tagstruct` (the structure number) and `tagstructobj` (the object reference).

`parent` (*struct key*) By default a structure is added as kid to the currently active structure. With the parent key one can choose another parent. The value is a structure number which must refer to an already existing, previously created structure. Such a structure number can for example be have been stored with `\tag_get:n`, but one can also use a label on the parent structure and then use `\property_ref:nn{tagpdfstruct-label}{tagstruct}` to retrieve it.

`firstkid` (*struct key*) If this key is used the structure is added at the left of the kids of the parent structure (if the structure is not stashed). This means that it will be the first kid of the structure (unless some later structure uses the key too).

`title` (*struct key*)
`title-o` (*struct key*) 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.

`alt` (*struct key*) This key inserts an `/Alt` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.

`actualtext` (*struct key*) This key inserts an `/ActualText` value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once. If it is empty, nothing will happen.

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

`ref` (*struct key*) 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` (*struct key*) 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` (*struct key*)
`AFref` (*struct key*)
`AFinline` (*struct key*)
`AFinline-o` (*struct key*)
`texsource` (*struct key*)
`mathml` (*struct key*)

These keys handle associated files in the structure element.

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

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.

`texsource` is a special variant of `AF-inline-o` which embeds the content as `.tex` source with the `/AFrelationship` key set to `/Source`. It also sets the `/Desc` key to a (currently) fix text.

`mathml` is a special variant of `AF-inline-o` which embeds the content as `.xml` file with the `/AFrelationship` key set to `/Supplement`. It also sets the `/Desc` key to a (currently) fix text.

The argument of `AF` is an object name referring an embedded file as declared for example with `\pdf_object_new:n` or with the l3pdffile module. `AF` expands its argument (this allows e.g. to use some variable for automatic numbering) and can be used more than once, to associate more than one file.

The argument of `AFref` is an object reference to an embedded file or a variable expanding to such a object reference in the format as you would get e.g. from `\pdf_object_ref_last:` or `\pdf_object_ref:n` (and which is different for the various engines!). The key allows to make use of anonymous objects. Like `AF` the `AFref` key expands its argument and can be used more than once, to associate more than one file. *It does not check if the reference is valid!*

The inline keys can be used only once per structure. Additional calls are ignored.

`attribute` (*struct key*) 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` (*struct key*) 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

```
role/new-attribute (setup-key) role/new-attribute = {⟨name⟩}{⟨Content⟩}
newattribute (deprecated)
```

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
 {
  role/new-attribute =
   {TH-col}{/O /Table /Scope /Column},
  role/new-attribute =
   {TH-row}{/O /Table /Scope /Row},
  }
```

`root-AF` (*setup key*)

```
root-AF = ⟨object name⟩
```

This key can be used in the setup command `\tagpdfsetup` and allows to add associated files to the root structure. Like `AF` it can be used more than once to add more than one file.

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-struct-code} {2025-06-27} {0.99s}
 {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.

```
⟨base⟩\int_new:N  \c@g__tag_struct_abs_int
⟨base⟩\int_gset:Nn \c@g__tag_struct_abs_int { 1 }
```

(*End of 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.

```
⟨*package⟩
\__tag_seq_new:N  \g__tag_struct_objR_seq
```

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

`\c__tag_struct_null_tl` In lua mode we have to test if the kids a null

```
\tl_const:Nn\c__tag_struct_null_tl {null}
```

(*End of definition for* `\c__tag_struct_null_tl`.)

`\g__tag_struct_cont_mc_prop` in generic mode it can happen after a page break that we have to inject into a structure sequence an additional mc after. We will store this additional info in a property. The key is the absolute mc num, the value the pdf directory.

```
\__tag_prop_new:N  \g__tag_struct_cont_mc_prop
```

(*End of definition for* `\g__tag_struct_cont_mc_prop`.)

`\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 {1}
```

(*End of 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}{StructTreeRoot}}
```

(*End of 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. It is already defined in `tagpdf-base`. The local temporary variable will hold the parent when we fetch it from the stack.

```
⟨/package⟩
⟨base⟩\tl_new:N  \g__tag_struct_stack_current_tl
⟨base⟩\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}
⟨*package⟩
\tl_new:N     \l__tag_struct_stack_parent_tmpa_tl
```

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

In luatex we will store the structure number as attribute.

```
\sys_if_engine_luatex:TF
 {
   \cs_new:Npn \__tag_struct_set_attribute:
    {
      \lua_now:e
      {
        tex.setattribute
         (
          "global",
          luatexbase.attributes.g__tag_structnum_attr,
          \g__tag_struct_stack_current_tl
         )
      }
    }
 }
```

```
{
  \cs_new_eq:NN \__tag_struct_set_attribute: \prg_do_nothing:
}
```

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_1_prop` for the root and `\g_@@_struct_N_prop`, $N >= 2$ 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,
    AF                 %pdf 2.0
  }

\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,                 % abbreviation
    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 of 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`
`\l__tag_struct_roletag_tl`
`\g__tag_struct_roletag_NS_tl`
`\l__tag_struct_parenttag_tl`
`\l__tag_struct_parenttag_NS_tl`

Use by the tag key to store the tag and the namespace. The `roletag` variables will hold locally rolemapping info needed for the parent-child checks. The `parenttag` variables allow to set the target role of the parent of stashed structures.

```
\tl_new:N \g__tag_struct_tag_tl
\tl_new:N \g__tag_struct_tag_NS_tl
\tl_new:N \l__tag_struct_roletag_tl
\tl_new:N \l__tag_struct_roletag_NS_tl
\tl_new:N \l__tag_struct_parenttag_tl
\tl_set:Nn \l__tag_struct_parenttag_tl {STASHED}
\tl_new:N \l__tag_struct_parenttag_NS_tl
\tl_set:Nn \l__tag_struct_parenttag_NS_tl {latex}
```

(*End of definition for* `\g__tag_struct_tag_tl` *and others.*)

`\g__tag_struct_label_num_prop`

This will hold for every structure label the associated structure number. The prop will allow to fill the /Ref key directly at the first compilation if the ref key is used.

```
\prop_new_linked:N \g__tag_struct_label_num_prop
```

(*End of definition for* `\g__tag_struct_label_num_prop`.)

`\l__tag_struct_elem_stash_bool`

This will keep track of the stash status

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

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

`\l__tag_struct_addkid_tl`

This decides if a structure kid is added at the left or right of the parent. The default is `right`.

```
\tl_new:N  \l__tag_struct_addkid_tl
\tl_set:Nn \l__tag_struct_addkid_tl {right}
```

(*End of definition for* `\l__tag_struct_addkid_tl`.)

3.2 Variables used by tagging code of basic elements

`\g__tag_struct_dest_num_prop`

This variable records for (some or all, not clear yet) destination names the related structure number to allow to reference them in a Ref. The key is the destination. It is currently used by the toc-tagging and sec-tagging code.

```
⟨/package⟩
⟨base⟩\prop_new_linked:N \g__tag_struct_dest_num_prop
⟨*package⟩
```

(*End of definition for* `\g__tag_struct_dest_num_prop`.)

\g__tag_struct_ref_by_dest_prop

This variable contains structures whose Ref key should be updated at the end to point to structured related with this destination. As this is probably need in other places too, it is not only a toc-variable. TODO: remove after 11/2024 release.

```
\prop_new_linked:N \g__tag_struct_ref_by_dest_prop
```

(*End of definition for* `\g__tag_struct_ref_by_dest_prop`.)

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}
      }
  }
⟨/package⟩
```

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

__tag_struct_prop_gput:nnn

The structure props must be filled in various places. For this we use a common command which also takes care of the debug package:

```
⟨*package | debug⟩
⟨package⟩\cs_new_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
⟨debug⟩\cs_set_protected:Npn \__tag_struct_prop_gput:nnn #1 #2 #3
 {
   \__tag_prop_gput:cnn
        { g__tag_struct_#1_prop }{#2}{#3}
⟨debug⟩\prop_gput:cnn { g__tag_struct_debug_#1_prop } {#2} {#3}
 }
\cs_generate_variant:Nn \__tag_struct_prop_gput:nnn {onn,nne,nee,nno}
⟨/package | debug⟩
```

(*End of definition for* `\__tag_struct_prop_gput:nnn`.)

4.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object is `@@/struct/1` 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.

```
⟨*package⟩
\tl_gset:Nn \g__tag_struct_stack_current_tl {1}
```

__tag_pdf_name_e:n

```
\cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
⟨/package⟩
```

(*End of definition for* `\__tag_pdf_name_e:n`.)

g__tag_struct_1_prop
g__tag_struct_kids_1_seq

```
⟨*package⟩
\__tag_prop_new:c { g__tag_struct_1_prop }
\__tag_new_output_prop_handler:n {1}
\__tag_seq_new:c  { g__tag_struct_kids_1_seq }

\__tag_struct_prop_gput:nne
  { 1 }
  { Type }
  { \pdf_name_from_unicode_e:n {StructTreeRoot} }

\__tag_struct_prop_gput:nne
  { 1 }
  { S }
  { \pdf_name_from_unicode_e:n {StructTreeRoot} }

\__tag_struct_prop_gput:nne
  { 1 }
  { tag }
  { {StructTreeRoot}{pdf} }

\__tag_struct_prop_gput:nne
  { 1 }
  { rolemap }
  { {StructTreeRoot}{pdf} }

\__tag_struct_prop_gput:nne
  { 1 }
  { parentrole }
  { {StructTreeRoot}{pdf} }

```

Namespaces are pdf 2.0. If the code moves into the kernel, the setting must be probably delayed.

```
\pdf_version_compare:NnF < {2.0}
 {
   \__tag_struct_prop_gput:nne
    { 1 }
    { Namespaces }
    { \pdf_object_ref:n { __tag/tree/namespaces } }
}
⟨/package⟩
```

In debug mode we have to copy the root manually as it is already setup:

```
⟨debug⟩\prop_new:c { g__tag_struct_debug_1_prop }
⟨debug⟩\seq_new:c  { g__tag_struct_debug_kids_1_seq }
⟨debug⟩\prop_gset_eq:cc { g__tag_struct_debug_1_prop }{ g__tag_struct_1_prop }
⟨debug⟩\prop_gremove:cn { g__tag_struct_debug_1_prop }{Namespaces}
```

(*End of definition for* `g__tag_struct_1_prop` *and* `g__tag_struct_kids_1_seq`.)

4.2 Adding the /ID key

Every structure gets automatically an ID which is currently simply calculated from the structure number.

`\__tag_struct_get_id:n`

```
⟨*package⟩
\cs_new:Npn \__tag_struct_get_id:n #1 %#1=struct num
  {
    (
     ID.
     \prg_replicate:nn
      { \int_abs:n{\g__tag_tree_id_pad_int - \tl_count:e { \int_to_arabic:n { #1 } }} }
      { 0 }
     \int_to_arabic:n { #1 }
    )
  }
```

(*End of definition for* `\__tag_struct_get_id:n`.)

4.3 Filling in the tag info

`\__tag_struct_set_tag_info:nnn` This adds or updates the tag info to a structure given by a number. We need also the original data, so we store both.

```
\pdf_version_compare:NnTF < {2.0}
 {
   \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
     %#1 structure number, #2 tag, #3 NS
     {
       \__tag_struct_prop_gput:nne
         { #1 }
         { S }
         { \pdf_name_from_unicode_e:n {#2}  } %
       \__tag_struct_prop_gput:nnn
         { #1 }
         { tag }
         { {#2} {} }
     }
 }
 {
   \cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
```

```
      {
        \__tag_struct_prop_gput:nne
          { #1 }
          { S }
          { \pdf_name_from_unicode_e:n {#2} } %
        \prop_get:NnNT \g__tag_role_NS_prop {#3} \l__tag_get_tmpc_tl
          {
            \__tag_struct_prop_gput:nne
              { #1 }
              { NS }
              { \l__tag_get_tmpc_tl } %
          }
        \__tag_struct_prop_gput:nnn
          { #1 }
          { tag }
          { {#2} {#3} }
      }
  }
\cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eoo}
```

(*End of definition for* `\__tag_struct_set_tag_info:nnn`.)

`\__tag_struct_get_role:nnNN` We also need a way to get the tag info needed for parent child check from parent structures. The tag info is stored as the value of the rolemap key, but for "transparent" structures we also have to look into parentrole key.

```
\cs_new_protected:Npn \__tag_struct_get_role:nnNN #1 #2 #3 #4
   %#1 :struct num,
   %#2 :rolemap or parentrole
   %#3 :tlvar for tag (rolemapped)
   %#4 :tlvar for NS  (rolemapped, so standard or empty or UNKNOWN)
    {
        \prop_get:cnNTF
          { g__tag_struct_#1_prop }
          { #2 }
          \l__tag_get_tmpc_tl
          {
            \tl_set:Ne #3{\exp_last_unbraced:No\use_i:nn  { \l__tag_get_tmpc_tl }}
            \tl_set:Ne #4{\exp_last_unbraced:No\use_ii:nn { \l__tag_get_tmpc_tl }}
          }
          {
            \tl_clear:N#3
            \tl_clear:N#4
          }
    }
\cs_generate_variant:Nn\__tag_struct_get_role:nnNN {enNN}
```

(*End of definition for* `\__tag_struct_get_role:nnNN`.)

4.4 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
__tag_struct_kid_mc_gput_right:ne

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. In generic mode the page break code will perhaps to have to insert an additional mcid after an existing one. For this we use a property list At first an auxiliary to write the MCID dict. This should normally be expanded!

```
\cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
  {
     <<
      /Type \c_space_tl /MCR \c_space_tl
      /Pg
        \c_space_tl
      \pdf_pageobject_ref:n { \property_ref:enn{mcid-#1}{tagabspage}{1} }
       /MCID \c_space_tl \property_ref:enn{mcid-#1}{tagmcid}{1}
     >>
  }
⟨/package⟩
⟨*package | debug⟩
⟨package⟩\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
⟨debug⟩\cs_set_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2
%#1 structure num, #2 MCID absnum%
  {
    \__tag_seq_gput_right:ce
      { g__tag_struct_kids_#1_seq }
      {
        \__tag_struct_mcid_dict:n {#2}
      }
⟨debug⟩     \seq_gput_right:cn
⟨debug⟩       { g__tag_struct_debug_kids_#1_seq }
⟨debug⟩       {
⟨debug⟩         MC~#2
⟨debug⟩       }
    \__tag_seq_gput_right:cn
      { g__tag_struct_kids_#1_seq }
      {
        \prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
      }
  }
⟨package⟩\cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {ne}
```

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

__tag_struct_kid_struct_gput_right:nn
__tag_struct_kid_struct_gput_right:ee

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

```
⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_struct_gput_right:nn #1 #2
%%#1 num of parent struct, #2 kid struct
  {
    \__tag_seq_gput_right:ce
      { g__tag_struct_kids_#1_seq }
      {
        \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
        }
⟨debug⟩     \seq_gput_right:cn
⟨debug⟩       { g__tag_struct_debug_kids_#1_seq }
⟨debug⟩       {
⟨debug⟩         Struct~#2
⟨debug⟩       }
  }
⟨package⟩\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {ee}
```

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

`\__tag_struct_kid_struct_gput_left:nn`
`\__tag_struct_kid_struct_gput_left:ee`

This commands adds a structure as kid one the left, so as first kid. We only need to record the object reference in the sequence.

```
⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_struct_gput_left:nn #1 #2
%%#1 num of parent struct, #2 kid struct
  {
    \__tag_seq_gput_left:ce
      { g__tag_struct_kids_#1_seq }
      {
        \pdf_object_ref_indexed:nn { __tag/struct }{ #2 }
      }
⟨debug⟩     \seq_gput_left:cn
⟨debug⟩       { g__tag_struct_debug_kids_#1_seq }
⟨debug⟩       {
⟨debug⟩         Struct~#2
⟨debug⟩       }
  }
⟨package⟩\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_left:nn {ee}
```

(*End of definition for* `\__tag_struct_kid_struct_gput_left:nn`.)

`\__tag_struct_kid_OBJR_gput_right:nnn`
`\__tag_struct_kid_OBJR_gput_right:eee`

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. The last argument is the page object reference

```
⟨package⟩\cs_new_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
⟨package⟩
⟨package⟩
⟨debug⟩\cs_set_protected:Npn\__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3
%%#1 num of parent struct,#2 obj reference,#3 page object reference
  {
    \pdf_object_unnamed_write:nn
      { dict }
      {
        /Type/OBJR/Obj~#2/Pg~#3
      }
    \__tag_seq_gput_right:ce
      { g__tag_struct_kids_#1_seq }
      {
        \pdf_object_ref_last:
      }
⟨debug⟩     \seq_gput_right:ce
⟨debug⟩       { g__tag_struct_debug_kids_#1_seq }
⟨debug⟩       {
```

```
⟨debug⟩            OBJR~reference
⟨debug⟩         }
  }
⟨/package | debug⟩
⟨*package⟩
\cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nnn { eee }
```

(*End of definition for* `\__tag_struct_kid_OBJR_gput_right:nnn`.)

`\__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. Change 2024-03-19: don't use a regex - that is slow.

```
\cs_new_protected:Npn\__tag_struct_exchange_kid_command:N #1 %#1 = seq var
  {
    \seq_gpop_left:NN #1 \l__tag_tmpa_tl
    \tl_replace_once:Nnn \l__tag_tmpa_tl
     {\__tag_mc_insert_mcid_kids:n}
     {\__tag_mc_insert_mcid_single_kids:n}
    \seq_gput_left:No #1 { \l__tag_tmpa_tl }
  }

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

(*End of 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
  {
    \bool_if:NF \g__tag_mode_lua_bool
     {
        \seq_clear:N \l__tag_tmpa_seq
        \seq_map_inline:cn { g__tag_struct_kids_#1_seq }
         { \seq_put_right:Ne \l__tag_tmpa_seq { ##1 } }
        %\seq_show:c { g__tag_struct_kids_#1_seq }
        %\seq_show:N \l__tag_tmpa_seq
        \seq_remove_all:Nn \l__tag_tmpa_seq {}
        %\seq_show:N \l__tag_tmpa_seq
        \seq_gset_eq:cN { g__tag_struct_kids_#1_seq } \l__tag_tmpa_seq
     }

    \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
         \sys_if_engine_luatex:TF
           {
             \__tag_struct_exchange_kid_command:c
              {g__tag_struct_kids_#1_seq}
```

check if we get null

```
              \tl_set:Ne\l__tag_tmpa_tl
                {\use:e{\seq_item:cn {g__tag_struct_kids_#1_seq} {1}}}
              \tl_if_eq:NNF\l__tag_tmpa_tl \c__tag_struct_null_tl
                {
                  \__tag_struct_prop_gput:nne
                    {#1}
                    {K}
                    {
                      \seq_item:cn
                        {
                          g__tag_struct_kids_#1_seq
                        }
                        {1}
                    }
                }
           }
           {
             \__tag_struct_prop_gput:nne
               {#1}
               {K}
               {
                 \seq_item:cn
                   {
                     g__tag_struct_kids_#1_seq
                   }
                   {1}
               }
           }
       } %
    }
    { %many kids, use an array
      \__tag_struct_prop_gput:nne
        {#1}
        {K}
        {
          [
            \seq_use:cn
              {
                g__tag_struct_kids_#1_seq
              }
              {
                \c_space_tl
              }
          ]
        }
    }
```

```
  }

```

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

4.5 Output of the object

`\__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. This is used a lot so should be rather fast.

```
\cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
  {
    \tl_clear:N #2
    \prop_map_inline:cn { g__tag_struct_#1_prop }
      {
```

Some keys needs the option to format the value, e.g. add brackets for an array, we also need the option to ignore some entries in the properties.

```
        \cs_if_exist_use:cTF {__tag_struct_format_##1:nnN}
          {
            {##1}{##2}#2
          }
          {
            \tl_put_right:Ne #2 { \c_space_tl/##1~##2 }
          }
      }
  }
```

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

`\__tag_struct_format_rolemap:nnN`
`\__tag_struct_format_parentrole:nnN`
`\__tag_struct_format_P:nnN`
`\__tag_struct_format_tag:nnN`

This three entries should not end in the PDF. Todo: check if the S/NS keys can be dropped and replaced by a processing of the tag key.

```
\cs_new:Nn\__tag_struct_format_rolemap:nnN{}
\cs_new:Nn\__tag_struct_format_parentrole:nnN{}
\cs_new:Nn\__tag_struct_format_tag:nnN{}
```

(*End of definition for* `\__tag_struct_format_rolemap:nnN` *and others.*)

`\__tag_struct_format_parentnum:nnN`

parent is a structure number and should expand to the object reference.

```
\cs_new_protected:Nn\__tag_struct_format_parentnum:nnN
  {
    \tl_put_right:Ne #3 { ~/P~\pdf_object_ref_indexed:nn { __tag/struct} { #2 } }
  }
```

(*End of definition for* `\__tag_struct_format_parentnum:nnN`.)

`\__tag_struct_format_Ref:nnN`

Ref is an array, we store values as a clist of commands that must be executed here, the formatting has to add also brackets.

```
\cs_new_protected:Nn\__tag_struct_format_Ref:nnN
  {
    \tl_put_right:Nn #3 { ~/#1~[ } %]
    \clist_map_inline:nn{ #2 }
     {
       ##1 #3
     }
    \tl_put_right:Nn #3
     { %[
       \c_space_tl]
     }
  }
```

(*End of definition for* `\__tag_struct_format_Ref:nnN`.)

`\__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
  {
    \prop_if_exist:cTF { g__tag_struct_#1_prop }
      {
```

It can happen that a structure is not used and so has not parent. Simply ignoring it is problematic as it is also recorded in the IDTree, so we make an artifact out of it.

```
        \prop_get:cnNF { g__tag_struct_#1_prop } {parentnum}\l__tag_tmpb_tl
          {
%            \prop_gput:cne { g__tag_struct_#1_prop } {P}
%              {\pdf_object_ref_indexed:nn { __tag/struct }{1}}
            \prop_gput:cne { g__tag_struct_#1_prop } {parentnum}{1}
            \prop_gput:cne { g__tag_struct_#1_prop } {S}{/Artifact}
            \seq_if_empty:cF {g__tag_struct_kids_#1_seq}
              {
                \msg_warning:nnee
                  {tag}
                  {struct-orphan}
                  { #1 }
                  {\seq_count:c{g__tag_struct_kids_#1_seq}}
              }
          }
        \__tag_struct_fill_kid_key:n { #1 }
        \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
        \pdf_object_write_indexed:nnne
           { __tag/struct }{ #1 }
           {dict}
           {
             \l__tag_tmpa_tl\c_space_tl
             /ID~\__tag_struct_get_id:n{#1}
           }

      }
      {
        \msg_error:nnn { tag } { struct-no-objnum } { #1}
      }
  }
```

(*End of 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:nne
          { 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 annotation/xform
 %#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_property_record:nn {@tag@objr@page@#2 }{ tagabspage }
        \__tag_struct_kid_OBJR_gput_right:eee
          {
            \l__tag_struct_stack_parent_tmpa_tl
          }
          {
            #1 %
          }
          {
            \pdf_pageobject_ref:n
              { \property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
          }
        % add the parent obj number to the parent tree:
        % the command always expands its arguments!
        \__tag_parenttree_add_objr:nn
          {
            #2
          }
          {
            \pdf_object_ref_indexed:nn
              { __tag/struct }{ \l__tag_struct_stack_parent_tmpa_tl }
          }
        % increase the int:
        \int_gincr:N \c@g__tag_parenttree_obj_int
      }
  }
```

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

`\__tag_struct_insert_annot_shipout:nnn` This command is similar to the previous one but is meant to be used at shipout (currently only sensible for luatex). To move the OBJR into the right structure it has to get the structure number additionally as argument. But as it is used at shipout it doesn't need a label to get the page reference but can use `\g_shipout_readonly_int`. It does *not* increase the parenttree integer (timing is wrong in lua), instead code using the command has to do it. See the lua code.

```
\cs_new_protected:Npn\__tag_struct_insert_annot_shipout:nnn #1#2#3
% #1 structnum, #2 object reference, #3 StructParentNum
 {
   \__tag_struct_kid_OBJR_gput_right:eee
     {
       #1
     }
     {
       #2
     }
     {
       \pdf_pageobject_ref:n
         { \int_use:N \g_shipout_readonly_int } %
     }
   % add the parent obj number to the parent tree:
   % the command always expands its arguments!
   \__tag_parenttree_add_objr:nn
     {
       #3
     }
     {
       \pdf_object_ref_indexed:nn
         { __tag/struct }{ #1 }
     }
  }
```

(*End of definition for* `\__tag_struct_insert_annot_shipout:nnn`.)

`\__tag_get_data_struct_tag:` this command allows `\tag_get:n` to get the current structure tag with the keyword `struct_tag`.

```
\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 of definition for* `\__tag_get_data_struct_tag:`.)

`\__tag_get_data_struct_id:` this command allows `\tag_get:n` to get the current structure id with the keyword `struct_id`.

```
\cs_new:Npn \__tag_get_data_struct_id:
  {
    \__tag_struct_get_id:n {\g__tag_struct_stack_current_tl}
  }
⟨/package⟩
```

(*End of definition for* `\__tag_get_data_struct_id:`.)

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

```
⟨*base⟩
\cs_new:Npn \__tag_get_data_struct_num:
  {
    \g__tag_struct_stack_current_tl
  }
⟨/base⟩
```

(*End of definition for* `\__tag_get_data_struct_num:`.)

`\__tag_get_data_struct_counter:` this command allows `\tag_get:n` to get the current state of the structure counter with the keyword `struct_counter`. By comparing the numbers it can be used to check the number of structure commands in a piece of code.

```
⟨*base⟩
\cs_new:Npn \__tag_get_data_struct_counter:
  {
    \int_use:N \c@g__tag_struct_abs_int
  }
⟨/base⟩
```

(*End of definition for* `\__tag_get_data_struct_counter:`.)

4.6 Commands for the parent-child checks

`\__tag_struct_check_parent_child_aux:nnnnN`

```
⟨*package⟩
\cs_new_protected:Npn \__tag_struct_check_parent_child_aux:nnnnN #1#2#3#4#5
  {
% #1 structure number of parent
% #2 key to use to retrieve role of parent (either rolemap or parentrole field)
% #3 structure number of parent
% #4 key to use to retrieve role of child (either rolemap or parentrole field)
% #5 tl for return value
```

get parent rolemap

```
    \__tag_struct_get_role:nnNN
      {#1}
      {#2}
      \l__tag_get_parent_tmpa_tl
      \l__tag_get_parent_tmpb_tl
```

get child rolemap

```
    \__tag_struct_get_role:nnNN
      {#3}
      {#4}
      \l__tag_get_child_tmpa_tl
      \l__tag_get_child_tmpb_tl
```

check

```
    \__tag_role_check_parent_child:ooooN
      { \l__tag_get_parent_tmpa_tl } % rolemapped from above
      { \l__tag_get_parent_tmpb_tl } % rolemapped from above
      { \l__tag_get_child_tmpa_tl } %
      { \l__tag_get_child_tmpb_tl } %
     #5
  }
```

(*End of definition for* `\__tag_struct_check_parent_child_aux:nnnnN`.)

`\__tag_struct_check_parent_child:nn` When comparing the relation between structures we use the structure numbers.

```
\cs_new_protected:Npn \__tag_struct_check_parent_child:nn #1 #2
% #1 structure number of parent
% #2 structure number of child. %
% This assumes that the fields rolemap/parentrole has already been filled.
  {
```

This records if logging is on

```
    \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
      {
        \prop_get:cnN{g__tag_struct_#1_prop}{tag}\l__tag_get_parent_tmpa_tl
        \prop_get:cnN{g__tag_struct_#2_prop}{tag}\l__tag_get_parent_tmpb_tl
        \msg_note:nnee
         { tag }
         { role-parent-child-check }
         {
           \quark_if_no_value:NTF \l__tag_get_parent_tmpa_tl
            {??}
            {
              \exp_last_unbraced:No\use_ii:nn
               { \l__tag_get_parent_tmpa_tl }
              :
             \exp_last_unbraced:No\use_i:nn
               { \l__tag_get_parent_tmpa_tl }
            }
         }
         {
           \quark_if_no_value:NTF \l__tag_get_parent_tmpb_tl
            {??}
            {
             \exp_last_unbraced:No\use_ii:nn
              { \l__tag_get_parent_tmpb_tl }
             :
             \exp_last_unbraced:No\use_i:nn
              { \l__tag_get_parent_tmpb_tl }
            }
         }
      }
    \__tag_struct_check_parent_child_aux:nnnnN
      {#1}
      {rolemap}
      {#2}
      {rolemap}
      \l__tag_parent_child_check_tl
```

if the return value is 7 we have to check against the parentrole field.

```
    \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
      {
        \__tag_struct_check_parent_child_aux:nnnnN
          {#1}
          {parentrole}
          {#2}
          {rolemap}
          \l__tag_parent_child_check_tl
      }
    \__tag_check_struct_forbidden_parent_child:onn
     {\l__tag_parent_child_check_tl}
     {#1}
     {#2}
  }
\cs_generate_variant:Nn \__tag_struct_check_parent_child:nn {oo}
```

(*End of definition for* `\__tag_struct_check_parent_child:nn`.)

`\__tag_struct_use_check_parent_child:nn` A similar command is needed if a structure is stashed and used. The child can be - a normal tag (e.g. H1) then rolemap = parentrole = H1pdf2 and we should test rolemap (parent) and rolemap (child) if = 7 parentrole (parent) and rolemap (child) That is the normal check above.

- Part/Div/Nonstruct then rolemap = Partpdf2 and parentrole = STASHEDlatex or target parentNS

If parentrole =STASHED we can't test if the child fits here. If parentrole is not STASHED, then would should test if target parent= rolemap (parent) or parentrole (parent) and if yet then test rolemap (child) against rolemap (parent) and if =7 rolemap(child) against parentrole(parent). that is again the normal check.

```
\cs_new_protected:Npn \__tag_struct_use_check_parent_child:nn #1 #2
% #1 structure number of parent
% #2 structure number of child. %
  {
    \__tag_struct_get_role:enNN
      {#2}
      {rolemap}
      \l__tag_get_child_tmpa_tl
      \l__tag_get_child_tmpb_tl
    \str_case:onTF { \l__tag_get_child_tmpa_tl }
      {
        {Part} {}
        {Div}  {}
        {NonStruct} {}
      }
      { %child=Part etc
        \__tag_struct_get_role:enNN
```

```
          {#2}
          {parentrole}
          \l__tag_get_child_tmpa_tl
          \l__tag_get_child_tmpb_tl
         \str_if_eq:noTF
          {STASHED}{\l__tag_get_child_tmpa_tl}
          {
            % warn about unknown relationship
          }
          {
            % test if
            \__tag_struct_get_role:enNN
             {#1}
             {parentrole}
             \l__tag_get_parent_tmpa_tl
             \l__tag_get_parent_tmpb_tl
            \tl_if_eq:NNTF\l__tag_get_parent_tmpa_tl \l__tag_get_child_tmpa_tl
             {
               \__tag_struct_check_parent_child:nn {#1}{#2}
             }
             {
               %warn that parent-tag was misused.
             }
          }
       }
       {
         %child not Part etc, normal parent child test.
         \__tag_struct_check_parent_child:nn {#1}{#2}
       }
  }
\cs_generate_variant:Nn  { \__tag_struct_use_check_parent_child:nn }{oo}
```

(*End of definition for* `\__tag_struct_use_check_parent_child:nn`.)

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.

This socket is used by the tag key. It allows to switch between the latex-tabs and the standard tags.

```
\socket_new:nn { tag/struct/tag }{1}
\socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
 {
   \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
    {
      \seq_set_split:Nne \l__tag_tmpa_seq { / }
        {#1/\l__tag_tmp_unused_tl}
    }
    {
      \seq_set_split:Nne \l__tag_tmpa_seq { / }
        {#1/}
    }
   \tl_gset:Ne \g__tag_struct_tag_tl   { \seq_item:Nn\l__tag_tmpa_seq {1} }
```

```
  \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
  \__tag_check_structure_tag:N \g__tag_struct_tag_tl
}

\socket_new_plug:nnn { tag/struct/tag }{ pdf-tags }
{
  \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
    {
     \seq_set_split:Nne \l__tag_tmpa_seq { / }
       {#1/\l__tag_tmp_unused_tl}
    }
    {
      \seq_set_split:Nne \l__tag_tmpa_seq { / }
        {#1/}
    }
  \tl_gset:Ne \g__tag_struct_tag_tl   { \seq_item:Nn\l__tag_tmpa_seq {1} }
  \tl_gset:Ne \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
  \__tag_role_get:ooNN
    { \g__tag_struct_tag_tl }
    { \g__tag_struct_tag_NS_tl}
    \l__tag_tmpa_tl
    \l__tag_tmpb_tl
  \tl_gset:Ne \g__tag_struct_tag_tl {\l__tag_tmpa_tl}
  \tl_gset:Ne \g__tag_struct_tag_NS_tl{\l__tag_tmpb_tl}
  \__tag_check_structure_tag:N \g__tag_struct_tag_tl
}
\socket_assign_plug:nn { tag/struct/tag } {latex-tags}
```

label (*struct key*)
stash (*struct key*)
parent (*struct key*)
firstkid (*struct key*)
tag (*struct key*)
title (*struct key*)
title-o (*struct key*)
alt (*struct key*)
actualtext (*struct key*)
lang (*struct key*)
ref (*struct key*)
E (*struct key*)
phoneme (*struct key*)

```
\keys_define:nn { __tag / struct }
  {
    label .code:n        =
      {
        \prop_gput:Nee\g__tag_struct_label_num_prop
          {#1}{\int_use:N \c@g__tag_struct_abs_int}
        \__tag_property_record:eo
          {tagpdfstruct-#1}
          { \c__tag_property_struct_clist }
      },
    stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
    parent .code:n       =
      {
        \bool_lazy_and:nnTF
          {
            \prop_if_exist_p:c { g__tag_struct_\int_eval:n {#1}_prop }
          }
          {
            \int_compare_p:nNn {#1}<{\c@g__tag_struct_abs_int}
          }
          { \tl_set:Ne \l__tag_struct_stack_parent_tmpa_tl { \int_eval:n {#1} } }
          {
            \msg_warning:nnee { tag } { struct-unknown }
              { \int_eval:n {#1} }
              { parent~key~ignored }
          }
      },
    parent .default:n    = {-1},
    parent-tag .code:n =
     {
        \prop_get:NeNTF \g__tag_role_tags_NS_prop {#1} \l__tag_tmp_unused_tl
          {
          \seq_set_split:Nne \l__tag_tmpa_seq { / }
            {#1/\l__tag_tmp_unused_tl}
          }
          {
            \seq_set_split:Nne \l__tag_tmpa_seq { / }
              {#1/}
          }
        \tl_set:Ne \l__tag_struct_parenttag_tl    { \seq_item:Nn\l__tag_tmpa_seq {1} }
        \tl_set:Ne \l__tag_struct_parenttag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
        \__tag_role_get:ooNN
          { \l__tag_struct_parenttag_tl }
          { \l__tag_struct_parenttag_NS_tl}
          \l__tag_tmpa_tl
          \l__tag_tmpb_tl
        \tl_set:No \l__tag_struct_parenttag_tl    {\l__tag_tmpa_tl}
        \tl_set:No \l__tag_struct_parenttag_NS_tl{\l__tag_tmpb_tl}
        \__tag_check_structure_tag:N \l__tag_struct_parenttag_tl
     },
 firstkid .code:n = { \tl_set:Nn \l__tag_struct_addkid_tl {left} },
    tag   .code:n        = % S property
      {
        \socket_use:nn { tag/struct/tag }{#1}
      },
    title .code:n        = % T property
      {
        \str_set_convert:Nnnn
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_struct_prop_gput:nne
          { \int_use:N \c@g__tag_struct_abs_int }
          { T }
          { <\l__tag_tmpa_str> }
      },
    title-o .code:n        = % T property
      {
        \str_set_convert:Nonn
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_struct_prop_gput:nne
          { \int_use:N \c@g__tag_struct_abs_int }
          { T }
          { <\l__tag_tmpa_str> }
      },
alt .code:n      = % Alt property
  {
   \tl_if_empty:oF{#1}
     {
      \str_set_convert:Noon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_struct_prop_gput:nne
        { \int_use:N \c@g__tag_struct_abs_int }
        { Alt }
        { <\l__tag_tmpa_str> }
     }
  },
alttext .meta:n = {alt=#1},
actualtext .code:n  = % ActualText property
  {
    \tl_if_empty:oF{#1}
      {
       \str_set_convert:Noon
         \l__tag_tmpa_str
         { #1 }
         { default }
         { utf16/hex }
       \__tag_struct_prop_gput:nne
         { \int_use:N \c@g__tag_struct_abs_int }
         { ActualText }
         { <\l__tag_tmpa_str>}
      }
  },
 phoneme .code:n  = % Phoneme property
  {
    \tl_if_empty:oF{#1}
      {
       \str_set_convert:Noon
         \l__tag_tmpa_str
         { #1 }
         { default }
         { utf16/hex }
       \__tag_struct_prop_gput:nne
         { \int_use:N \c@g__tag_struct_abs_int }
         { Phoneme }
         { <\l__tag_tmpa_str>}
      }
  },
lang .code:n        = % Lang property
  {
    \__tag_struct_prop_gput:nne
      { \int_use:N \c@g__tag_struct_abs_int }
      { Lang }
      { (#1) }
  },
}
```

Ref is rather special as it values are often known only at the end of the document. It therefore stores it values as clist of commands which are executed at the end of the document, when the structure elements are written.

__tag_struct_Ref_obj:nN
__tag_struct_Ref_label:nN
__tag_struct_Ref_dest:nN
__tag_struct_Ref_num:nN

this commands are helper commands that are stored as clist in the Ref key of a structure. They are executed when the structure elements are written in `\__tag_struct_write_obj`. They are used in `\__tag_struct_format_Ref`. They allow to add a Ref by object reference, label, destname and structure number

```
\cs_new_protected:Npn \__tag_struct_Ref_obj:nN #1 #2 %#1 a object reference
  {
    \tl_put_right:Ne#2
      {
        \c_space_tl#1
      }
  }

\cs_new_protected:Npn \__tag_struct_Ref_label:nN #1 #2 %#1 a label
  {
    \prop_get:NnNTF \g__tag_struct_label_num_prop {#1} \l__tag_tmpb_tl
      {
        \tl_put_right:Ne#2
          {
            \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
          }
      }
      {
        \msg_warning:nnn {tag}{struct-Ref-unknown}{Label~'#1'}
      }
  }
\cs_new_protected:Npn \__tag_struct_Ref_dest:nN #1 #2 %#1 a dest name
  {
    \prop_get:NnNTF \g__tag_struct_dest_num_prop {#1} \l__tag_tmpb_tl
      {
        \tl_put_right:Ne#2
          {
            \c_space_tl\tag_struct_object_ref:e{ \l__tag_tmpb_tl }
          }
      }
      {
        \msg_warning:nnn {tag}{struct-Ref-unknown}{Destination~'#1'}
      }
  }
\cs_new_protected:Npn \__tag_struct_Ref_num:nN #1 #2 %#1 a structure number
  {
    \tl_put_right:Ne#2
      {
        \c_space_tl\tag_struct_object_ref:e{ #1 }
      }
  }

```

(*End of definition for* `\__tag_struct_Ref_obj:nN` *and others.*)

`ref` (*struct key*)
`E` (*struct key*)

```
\keys_define:nn { __tag / struct }
  {
    ref .code:n        = % ref property
      {
        \clist_map_inline:on {#1}
          {
            \tag_struct_gput:nne
              {\int_use:N \c@g__tag_struct_abs_int}{ref_label}{ ##1 }
          }
      },
    E .code:n        = % E property
      {
        \str_set_convert:Nnon
          \l__tag_tmpa_str
          { #1 }
          { default }
          { utf16/hex }
        \__tag_struct_prop_gput:nne
          { \int_use:N \c@g__tag_struct_abs_int }
          { E }
          { <\l__tag_tmpa_str> }
      },
  }
```

`AF` (*struct key*)
`AFref` (*struct key*)
`AFinline` (*struct key*)
`AFinline-o` (*struct key*)
`texsource` (*struct key*)
`mathml` (*struct key*)

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. AF/AFref is an array and can be used more than once, so we store it in a tl. which is expanded. AFinline currently uses the fix extension txt. texsource is a special variant which creates a tex-file, it expects a tl-var as value (e.g. from math grabbing)

`\g__tag_struct_AFobj_int` This variable is used to number the AF-object names

```
\int_new:N\g__tag_struct_AFobj_int
```

(*End of definition for* `\g__tag_struct_AFobj_int`.)

```
\cs_generate_variant:Nn \pdffile_embed_stream:nnN {neN}
\cs_new_protected:Npn \__tag_struct_add_inline_AF:nn #1 #2
% #1 content, #2 extension
 {
   \tl_if_empty:nF{#1}
     {
      \group_begin:
      \int_gincr:N \g__tag_struct_AFobj_int
      \pdffile_embed_stream:neN
        {#1}
        {tag-AFfile\int_use:N\g__tag_struct_AFobj_int.#2}
        \l__tag_tmpa_tl
        \__tag_struct_add_AF:ee
          { \int_use:N \c@g__tag_struct_abs_int }
          { \l__tag_tmpa_tl }
        \__tag_struct_prop_gput:nne
          { \int_use:N \c@g__tag_struct_abs_int }
          { AF }
          {
            [
              \tl_use:c
               { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
            ]
          }
       \group_end:
      }
 }

\cs_generate_variant:Nn \__tag_struct_add_inline_AF:nn {on}
\cs_new_protected:Npn \__tag_struct_add_AF:nn #1 #2
% #1 struct num #2 object reference
  {
     \tl_if_exist:cTF
       {
         g__tag_struct_#1_AF_tl
       }
       {
         \tl_gput_right:ce
           { g__tag_struct_#1_AF_tl }
           {  \c_space_tl #2 }
       }
       {
          \tl_new:c
            { g__tag_struct_#1_AF_tl }
          \tl_gset:ce
            { g__tag_struct_#1_AF_tl }
            { #2 }
       }
  }
\cs_generate_variant:Nn \__tag_struct_add_AF:nn {en,ee}
\keys_define:nn { __tag / struct }
 {
    AF .code:n        = % AF property
      {
        \pdf_object_if_exist:eTF {#1}
          {
            \__tag_struct_add_AF:ee
             { \int_use:N \c@g__tag_struct_abs_int }{\pdf_object_ref:e {#1}}
            \__tag_struct_prop_gput:nne
             { \int_use:N \c@g__tag_struct_abs_int }
             { AF }
             {
               [
                 \tl_use:c
                   { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
               ]
             }
          }
          {
             % message?
          }
      },
    AFref .code:n        = % AF property
```

```
      {
        \tl_if_empty:eF {#1}
          {
            \__tag_struct_add_AF:ee { \int_use:N \c@g__tag_struct_abs_int }{#1}
            \__tag_struct_prop_gput:nne
              { \int_use:N \c@g__tag_struct_abs_int }
              { AF }
              {
                [
                  \tl_use:c
                  { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
                ]
              }
          }
      },
    ,AFinline .code:n =
      {
        \__tag_struct_add_inline_AF:nn {#1}{txt}
      }
    ,AFinline-o .code:n =
      {
        \__tag_struct_add_inline_AF:on {#1}{txt}
      }
    ,texsource .code:n =
     {
       \group_begin:
       \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(TeX~source)}
       \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Source }
       \__tag_struct_add_inline_AF:on {#1}{tex}
       \group_end:
     }
    ,mathml .code:n =
      {
       \group_begin:
       \pdfdict_put:nnn { l_pdffile/Filespec } {Desc}{(mathml~representation)}
       \pdfdict_put:nnn { l_pdffile/Filespec }{AFRelationship} { /Supplement }
       \pdfdict_put:nne { l_pdffile }{Subtype}
          { \pdf_name_from_unicode_e:n{application/mathml+xml} }
       \__tag_struct_add_inline_AF:on {#1}{xml}
       \group_end:
     }
 }
```

`root-AF` (*setup key*) The root structure can take AF keys too, so we provide a key for it. This key is used with `\tagpdfsetup`, not in a structure!

```
\keys_define:nn { __tag / setup }
  {
    root-AF .code:n =
     {
        \pdf_object_if_exist:nTF {#1}
          {
            \__tag_struct_add_AF:ee { 1 }{\pdf_object_ref:n {#1}}
            \__tag_struct_prop_gput:nne
             { 1 }
```

```
              { AF }
              {
                [
                  \tl_use:c
                    { g__tag_struct_1_AF_tl }
                ]
              }
           }
           {

           }
       },
  }
```

ot-supplemental-file (*setup key*) This key allows to add a file as root-AF with relationship Supplement. This is typically need to add a css or an html

```
\keys_define:nn { __tag / setup }
  {
    root-supplemental-file .code:n =
     {
       \group_begin:
       \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
       \int_gincr:N \g__tag_unique_cnt_int
       \pdffile_embed_file:eee
        {#1}
        {#1}
        {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
       \keys_set:nn
        {__tag / setup}
        {root-AF={__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}}
       \group_end:
     }
  }
```

og-supplemental-file (*setup key*) This key allows to add a file as AF with relationship Supplement to the Catalog. This is typically need to add a css or an html.

```
\keys_define:nn { __tag / setup }
  {
    catalog-supplemental-file .code:n =
     {
       \group_begin:
       \pdfdict_put:nnn {l_pdffile/Filespec} {AFRelationship}{/Supplement}
       \int_gincr:N \g__tag_unique_cnt_int
       \pdffile_embed_file:eee
        {#1}
        {#1}
        {__tag_latex_css_\int_use:N\g__tag_unique_cnt_int}
       \pdfmanagement_add:nne
        {Catalog}
        {AF}
        {\pdf_object_ref:e{__tag_latex_css_\int_use:N\g__tag_unique_cnt_int }}
       \group_end:
     }
  }
```

6 User commands

We allow to set a language by default

\l__tag_struct_lang_tl

```
\tl_new:N \l__tag_struct_lang_tl
⟨/package⟩
```

(*End of definition for* \l__tag_struct_lang_tl.)

\tag_struct_begin:n
\tag_struct_end:

```
⟨base⟩\cs_new_protected:Npn \tag_struct_begin:n #1 {\int_gincr:N \c@g__tag_struct_abs_int}
⟨base⟩\cs_new_protected:Npn \tag_struct_end:{}
⟨base⟩\cs_new_protected:Npn \tag_struct_end:n{}
⟨*package | debug⟩
⟨package⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
⟨debug⟩\cs_set_protected:Npn \tag_struct_begin:n #1 %#1 key-val
  {
⟨package⟩\__tag_check_if_active_struct:T
⟨debug⟩\__tag_check_if_active_struct:TF
      {
        \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 }
⟨debug⟩        \prop_new:c { g__tag_struct_debug_\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}
⟨debug⟩        \seq_new:c { g__tag_struct_debug_kids_\int_eval:n {\c@g__tag_struct_abs_int}_
          \pdf_object_new_indexed:nn { __tag/struct }
            { \c@g__tag_struct_abs_int }
         \__tag_struct_prop_gput:nnn
            { \int_use:N \c@g__tag_struct_abs_int }
            { Type }
            { /StructElem }
         \tl_if_empty:NF \l__tag_struct_lang_tl
           {
             \__tag_struct_prop_gput:nne
              { \int_use:N \c@g__tag_struct_abs_int }
              { Lang }
              { (\l__tag_struct_lang_tl) }
           }
         \__tag_struct_prop_gput:nnn
            { \int_use:N \c@g__tag_struct_abs_int }
            { Type }
            { /StructElem }

        \tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
        \keys_set:nn { __tag / struct} { #1 }

        \__tag_struct_set_tag_info:eoo
          { \int_use:N \c@g__tag_struct_abs_int }
          { \g__tag_struct_tag_tl }
          { \g__tag_struct_tag_NS_tl }
        \__tag_check_structure_has_tag:n { \int_use:N \c@g__tag_struct_abs_int }
```

The structure number of the parent is either taken from the stack or has been set with the parent key.

```
        \int_compare:nNnT { \l__tag_struct_stack_parent_tmpa_tl } = { -1 }
          {
            \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
        \__tag_role_get:ooNN
          { \g__tag_struct_tag_tl }
          { \g__tag_struct_tag_NS_tl }
          \l__tag_struct_roletag_tl
          \l__tag_struct_roletag_NS_tl
```

We push the role tag on the stack:

```
        \seq_gpush:Ne \g__tag_struct_tag_stack_seq
          {{\g__tag_struct_tag_tl}{\l__tag_struct_roletag_tl}}
        \tl_gset:NV   \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
        \__tag_struct_set_attribute:
        %\seq_show:N   \g__tag_struct_stack_seq
```

the rolemapped role and its NS are stored in the rolemap key.

```
         \__tag_struct_prop_gput:nne
           { \int_use:N \c@g__tag_struct_abs_int }
           { rolemap }
           {
             {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
           }
```

If the role is one of Part, Div, NonStruct we have to (sometimes) retrieve the "real" parent for the parent/child test. The role of this real parent is stored in the key `parentrole`. If the current structure is stashed we use UNKNOWN as real parent if the current structure is rolemapped to Part, Div or NonStruct so that the children can detect that no reliable check is possible. For structures that are not rolemapped to Part, Div, NonStruct, `parentrole` and `rolemap` are always equal.

```
        \str_case:onTF { \l__tag_struct_roletag_tl }
         {
           {Part} {}
           {Div}  {}
           {NonStruct} {}
         }
         {
           \bool_if:NTF \l__tag_struct_elem_stash_bool
           {
             \__tag_struct_prop_gput:nne
               { \int_use:N \c@g__tag_struct_abs_int }
               { parentrole }
               {
                 {\l__tag_struct_parenttag_tl}{\l__tag_struct_parenttag_NS_tl}
               }
```

```
            }
            {
              \prop_get:cnNT
                { g__tag_struct_ \l__tag_struct_stack_parent_tmpa_tl _prop }
                { parentrole }
                \l__tag_get_tmpc_tl
                {
                  \__tag_struct_prop_gput:nno
                    { \int_use:N \c@g__tag_struct_abs_int }
                    { parentrole }
                    {
                      \l__tag_get_tmpc_tl
                    }
                }
            }
          }
          {
             \__tag_struct_prop_gput:nne
               { \int_use:N \c@g__tag_struct_abs_int }
               { parentrole }
               {
                 {\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
               }
          }
        \bool_if:NF
          \l__tag_struct_elem_stash_bool
          {
```

check if the tag can be used inside the parent. It only makes sense, if the structure is actually used here, so it is guarded by the stash boolean.

```
             \socket_use:nn{tag/check/parent-child}
              {
                \__tag_struct_check_parent_child:oo
                  { \l__tag_struct_stack_parent_tmpa_tl }
                  { \int_use:N \c@g__tag_struct_abs_int }
              }
```

Set the Parent structure number.

```
             \__tag_struct_prop_gput:nne
               { \int_use:N \c@g__tag_struct_abs_int }
               { parentnum }
               {
                 \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
             \use:c { __tag_struct_kid_struct_gput_ \l__tag_struct_addkid_tl :ee }
                { \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}
          }
```

the debug mode stores in second prop and replaces value with more suitable ones. (If the structure is updated later this gets perhaps lost, but well ...) This must be done outside of the stash boolean.

```
⟨debug⟩           \prop_gset_eq:cc
⟨debug⟩             { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
⟨debug⟩             { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
⟨debug⟩           \prop_gput:cne
⟨debug⟩             { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
⟨debug⟩             { parentnum }
⟨debug⟩             {
⟨debug⟩               \bool_if:NTF \l__tag_struct_elem_stash_bool
⟨debug⟩                 {no~parent:~stashed}
⟨debug⟩                 {
⟨debug⟩                   \l__tag_struct_stack_parent_tmpa_tl\c_space_tl =~
⟨debug⟩                   \prop_item:cn{ g__tag_struct_\l__tag_struct_stack_parent_tmpa_tl _p
⟨debug⟩                 }
⟨debug⟩             }
⟨debug⟩           \prop_gput:cne
⟨debug⟩             { g__tag_struct_debug_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
⟨debug⟩             { NS }
⟨debug⟩             { \g__tag_struct_tag_NS_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}
⟨debug⟩ \__tag_debug_struct_begin_insert:n { #1 }
          \group_end:
        }
⟨debug⟩{ \__tag_debug_struct_begin_ignore:n { #1 }}
  }
⟨package⟩\cs_set_protected:Nn \tag_struct_end:
⟨debug⟩\cs_set_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
⟨package⟩\__tag_check_if_active_struct:T
⟨debug⟩\__tag_check_if_active_struct:TF
      {
        \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:No  \g__tag_struct_stack_current_tl { \l__tag_tmpa_tl }
            \__tag_struct_set_attribute:
          }
          {
            \__tag_check_no_open_struct:
          }
        \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
          {
            \tl_gset:Ne \g__tag_struct_tag_tl
              { \exp_last_unbraced:No\use_i:nn { \l__tag_tmpa_tl } }
            \prop_get:NoNT\g__tag_role_tags_NS_prop { \g__tag_struct_tag_tl} \l__tag_tmpa_tl
             {
               \tl_gset:Ne \g__tag_struct_tag_NS_tl { \l__tag_tmpa_tl }
             }
          }
⟨debug⟩\__tag_debug_struct_end_insert:
      }
⟨debug⟩{\__tag_debug_struct_end_ignore:}
  }

\cs_set_protected:Npn \tag_struct_end:n #1
 {
⟨debug⟩     \__tag_check_if_active_struct:T{\__tag_debug_struct_end_check:n{#1}}
   \tag_struct_end:
 }
⟨/package | debug⟩
```

(*End of definition for* `\tag_struct_begin:n` *and* `\tag_struct_end:`. *These functions are documented on page 110.*)

`\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.

```
⟨base⟩\cs_new_protected:Npn \tag_struct_use:n #1 {}
⟨*package | debug⟩
\cs_set_protected:Npn \tag_struct_use:n #1 %#1 is the label
  {
    \__tag_check_if_active_struct:T
      {
        \prop_if_exist:cTF
          { g__tag_struct_\property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop } %
          {
            \__tag_check_struct_used:n {#1}
            \tl_set:Ne \l__tag_get_child_tmpa_tl
              { \property_ref:enn{tagpdfstruct-#1}{tagstruct}{1} }
```

add the label structure as kid to the current structure (can be the root)

```
            \__tag_struct_kid_struct_gput_right:ee
              { \g__tag_struct_stack_current_tl }
              { \l__tag_get_child_tmpa_tl }
```

add the current structure to the labeled one as parents

```
            \__tag_prop_gput:cne
              { g__tag_struct_ \l__tag_get_child_tmpa_tl _prop }
              { parentnum }
              {
                \g__tag_struct_stack_current_tl
              }
```

debug code

```
⟨debug⟩            \prop_gput:cne
⟨debug⟩              { g__tag_struct_debug_ \l__tag_get_child_tmpa_tl _prop }
⟨debug⟩              { parentnum }
⟨debug⟩              {
⟨debug⟩                \g__tag_struct_stack_current_tl\c_space_tl=~
⟨debug⟩                    \g__tag_struct_tag_tl
⟨debug⟩                  }
```

check if the tag is allowed as child. If the tag of the child after rolemapping is *not* one of Part, Div, NonStruct, then the parentrole field will be identically to the rolemap field and can be used for a check. Otherwise the parentrole will contain latex:STASHED (if not changed with the `parent-tag` key when the structure was stashed) and will produce a warning.

```
              \socket_use:nn{tag/check/parent-child}
               {
                 \__tag_struct_use_check_parent_child:oo
                   { \g__tag_struct_stack_current_tl }
                   { \l__tag_get_child_tmpa_tl }
               }
             }
             {
               \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
             }
        }
    }
⟨/package | debug⟩
```

(*End of definition for* `\tag_struct_use:n`. *This function is documented on page 110.*)

`\tag_struct_use_num:n` This command allows to use a stashed structure in another place. differently to the previous command it doesn't use a label but directly a structure number to find the parent. TODO: decide how it should be guarded. Probably by the struct-check.

```
⟨base⟩\cs_new_protected:Npn \tag_struct_use_num:n #1 {}
⟨*package | debug⟩
\cs_set_protected:Npn \tag_struct_use_num:n #1 %#1 is structure number
  {
    \__tag_check_if_active_struct:T
      {
        \prop_if_exist:cTF
          { g__tag_struct_#1_prop } %
          {
            \prop_get:cnNT
              {g__tag_struct_#1_prop}
              {parentnum}
              \l__tag_tmpa_tl
              {
                \msg_warning:nnn { tag } {struct-used-twice} {#1}
              }
```

add the #1 structure as kid to the current structure (can be the root)

```
            \__tag_struct_kid_struct_gput_right:ee
              { \g__tag_struct_stack_current_tl }
              { #1 }
```

add the current structure to #1 as parent

```
             \__tag_struct_prop_gput:nne
              { #1 }
              { parentnum }
              {
                \g__tag_struct_stack_current_tl
```

```
                  }
⟨debug⟩              \prop_gput:cne
⟨debug⟩                { g__tag_struct_debug_#1_prop }
⟨debug⟩                { parentnum }
⟨debug⟩                {
⟨debug⟩                  \g__tag_struct_stack_current_tl\c_space_tl=~
⟨debug⟩                  \g__tag_struct_tag_tl
⟨debug⟩                }
```

check if the tag is allowed as child.

```
                \socket_use:nn{tag/check/parent-child}
                 {
                   \__tag_struct_use_check_parent_child:oo
                     {\g__tag_struct_stack_current_tl}
                     {#1}
                 }
             }
             {
               \msg_warning:nnn{ tag }{struct-label-unknown}{#1}
             }
        }
   }
⟨/package | debug⟩
```

(*End of definition for* `\tag_struct_use_num:n`. *This function is documented on page 110.*)

`\tag_struct_object_ref:n` This is a command that allows to reference a structure. The argument is the number which can be get for the current structure with `\tag_get:n{struct_num}` TODO check if it should be in base too.

```
⟨*package⟩
\cs_new:Npn \tag_struct_object_ref:n #1
 {
   \pdf_object_ref_indexed:nn {__tag/struct}{ #1 }
 }
\cs_generate_variant:Nn \tag_struct_object_ref:n {e}
⟨/package⟩
```

(*End of definition for* `\tag_struct_object_ref:n`. *This function is documented on page 110.*)

`\tag_struct_gput:nnn` This is a command that allows to update the data of a structure. This often can't done simply by replacing the value, as we have to preserve and extend existing content. We use therefore dedicated functions adjusted to the key in question. The first argument is the number of the structure, the second a keyword referring to a function, the third the value. Currently the existing keywords are mostly related to the `Ref` key (an array). The keyword `ref` takes as value an explicit object reference to a structure. The keyword `ref_label` expects as value a label name (from a label set in a `\tagstructbegin` command). The keyword `ref_dest` expects a destination name set with `\MakeLinkTarget`. It then will refer to the structure in which this `\MakeLinkTarget` was used. The keyword `ref_num` expects a structure number. At last there is the keyword `attribute` which allows to add or extend the `/A` key of the structure. The value is the content of one attribute dictionary, so for example `/O /Layout /BBox [10 10 50 50]`. The content is stored in an object and the object reference is than added to the `/A`.

```
⟨base⟩\cs_new_protected:Npn \tag_struct_gput:nnn #1 #2 #3{}
```

```
⟨*package⟩
\cs_set_protected:Npn \tag_struct_gput:nnn #1 #2 #3
 {
   \cs_if_exist_use:cF {__tag_struct_gput_data_#2:nn}
    { %warning??
      \use_none:nn
    }
    {#1}{#3}
 }
\cs_generate_variant:Nn \tag_struct_gput:nnn {ene,nne}
⟨/package⟩
```

(*End of definition for* `\tag_struct_gput:nnn`. *This function is documented on page 111.*)

\__tag_struct_gput_data_ref_aux:nnn

```
⟨*package⟩
\cs_new_protected:Npn \__tag_struct_gput_data_ref_aux:nnn #1 #2 #3
  % #1 receiving struct num, #2 key word #3 value
   {
     \prop_get:cnNTF
       { g__tag_struct_#1_prop }
       {Ref}
       \l__tag_get_tmpc_tl
       {
         \tl_put_right:No \l__tag_get_tmpc_tl
           {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
       }
       {
         \tl_set:No \l__tag_get_tmpc_tl
           {\cs:w __tag_struct_Ref_#2:nN \cs_end: {#3},}
       }
       \__tag_struct_prop_gput:nno
         { #1 }
         { Ref }
         { \l__tag_get_tmpc_tl }
    }
\cs_new_protected:Npn \__tag_struct_gput_data_ref:nn #1 #2
  {
    \__tag_struct_gput_data_ref_aux:nnn {#1}{obj}{#2}
  }
\cs_new_protected:Npn \__tag_struct_gput_data_ref_label:nn #1 #2
  {
    \__tag_struct_gput_data_ref_aux:nnn {#1}{label}{#2}
  }
\cs_new_protected:Npn \__tag_struct_gput_data_ref_dest:nn #1 #2
  {
    \__tag_struct_gput_data_ref_aux:nnn {#1}{dest}{#2}
  }
\cs_new_protected:Npn \__tag_struct_gput_data_ref_num:nn #1 #2
  {
    \__tag_struct_gput_data_ref_aux:nnn {#1}{num}{#2}
  }

\cs_generate_variant:Nn \__tag_struct_gput_data_ref:nn {ee,no}
```

(*End of definition for* `\__tag_struct_gput_data_ref_aux:nnn`.)

`\__tag_struct_gput_data_attribute:nn`

```
\cs_new_protected:Npn \__tag_struct_gput_data_attribute:nn #1 #2
  {
    \pdf_object_unnamed_write:nn {dict} {#2}
    \prop_get:cnNTF { g__tag_struct_#1_prop }{A} \l__tag_tmpa_tl
      {
        \tl_remove_once:Nn\l__tag_tmpa_tl{[}
        \tl_remove_once:Nn\l__tag_tmpa_tl{]}
        \__tag_prop_gput:cne { g__tag_struct_#1_prop }
         { A }
         {
           [ \l__tag_tmpa_tl \c_space_tl \pdf_object_ref_last: ]
         }
      }
      {
        \__tag_prop_gput:cne { g__tag_struct_#1_prop }
         { A }
         { \pdf_object_ref_last: }
      }
  }
```

(*End of definition for* `\__tag_struct_gput_data_attribute:nn`.)

`\tag_struct_insert_annot:nn`
`\tag_struct_insert_annot:ee`
`\tag_struct_insert_annot:ee`
`\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,ee}
\cs_new:Npn \tag_struct_parent_int: {\int_use:c { c@g__tag_parenttree_obj_int }}

⟨/package⟩

```

(*End of definition for* `\tag_struct_insert_annot:nn` *and* `\tag_struct_parent_int:`. *These functions are documented on page 110.*)

7 Attributes and attribute classes

```
⟨*header⟩
\ProvidesExplPackage {tagpdf-attr-code} {2025-06-27} {0.99s}
  {part of tagpdf - code related to attributes and attribute classes}
```

```
⟨/header⟩
```

7.1 Variables

\g__tag_attr_entries_prop
\g__tag_attr_class_used_prop
\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_prop` will hold the attributes which have been used as class name. `\l_@@_attr_value_tl` is used to build the attribute array or key. Every time 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
\prop_new_linked:N \g__tag_attr_class_used_prop
\tl_new:N   \l__tag_attr_value_tl
\prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes
```

This seq is currently kept for compatibility with the table code.

```
\seq_new:N\g__tag_attr_class_used_seq
```

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

7.2 Commands and keys

__tag_attr_new_entry:nn
role/new-attribute (setup-key)
newattribute (deprecated)

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

TODO: consider to put them directly in the ClassMap, that is perhaps more effective.

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

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

\cs_generate_variant:Nn \__tag_attr_new_entry:nn {ee}
\keys_define:nn { __tag / setup }
  {
    role/new-attribute .code:n =
      {
        \__tag_attr_new_entry:nn #1
      }
```

deprecated name

```
   ,newattribute .code:n =
      {
        \__tag_attr_new_entry:nn #1
      },
  }
```

(*End of definition for* `\__tag_attr_new_entry:nn`, `role/new-attribute (setup-key)`, *and* `newattribute (deprecated)`. *These functions are documented on page 113.*)

`attribute-class` (*struct key*) 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:Ne \l__tag_tmpa_clist { #1 }
       \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

we convert the names into pdf names with slash

```
       \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
         {
           \pdf_name_from_unicode_e:n {##1}
         }
       \seq_map_inline:Nn \l__tag_tmpa_seq
         {
           \prop_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmpa_tl
             {
               \msg_error:nnn { tag } { attr-unknown } { ##1 }
             }
           \prop_gput:Nnn\g__tag_attr_class_used_prop { ##1} {}
         }
       \tl_set:Ne \l__tag_tmpa_tl
         {
           \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{[}
           \seq_use:Nn \l__tag_tmpa_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_struct_prop_gput:nne
             { \int_use:N \c@g__tag_struct_abs_int }
             { C }
             { \l__tag_tmpa_tl }
          %\prop_show:c  { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
         }
     }
  }
```

`attribute` (*struct key*)

```
\keys_define:nn { __tag / struct }
  {
    attribute .code:n  = % A property (attribute, value currently a dictionary)
      {
        \clist_set:Ne          \l__tag_tmpa_clist { #1 }
        \clist_if_empty:NF \l__tag_tmpa_clist
         {
            \seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
```

we convert the names into pdf names with slash

```
           \seq_set_map_e:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
             {
               \pdf_name_from_unicode_e:n {##1}
             }
            \tl_set:Ne \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_get:NnNF \g__tag_attr_entries_prop {##1}\l__tag_tmp_unused_tl
                  {
                    \msg_error:nnn { tag } { attr-unknown } { ##1 }
                  }
                \prop_get:NnNF \g__tag_attr_objref_prop {##1}\l__tag_tmpa_tl
                  {%\prop_show:N \g__tag_attr_entries_prop
                    \pdf_object_unnamed_write:ne
                      { dict }
                      {
                        \prop_item:Nn\g__tag_attr_entries_prop {##1}
                      }
                    \prop_gput:Nne \g__tag_attr_objref_prop {##1} {\pdf_object_ref_last:}
                  }
                \tl_put_right:Ne \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:Ne \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_struct_prop_gput:nne
              { \int_use:N \c@g__tag_struct_abs_int }
              { A }
              { \l__tag_attr_value_tl }
        }
    },
  }
⟨/package⟩
```

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

```
⟨@@=tag⟩
⟨*luatex⟩
\ProvidesExplFile {tagpdf-luatex.def} {2025-06-27} {0.99s}
  {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_gput_left: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_prop_new_linked:N #1
  {
    \prop_new_linked: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"] = "\lua_escape:n{#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") }
  }
```

this inserts on the right of the lua table, but as the lua table is not used for kids this is ignored for now.

```
\cs_set_protected:Npn \__tag_seq_gput_left:Nn #1 #2
  {
    \seq_gput_left: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.sprint(\int_use:N\c_document_cctab,ltx.__tag.tables['#1'][#2]) }
  }

\cs_set:Npn \__tag_prop_item:cn #1 #2
  {
    \lua_now:e { tex.sprint(\int_use:N\c_document_cctab,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 of definition for* `\__tag_prop_new:N` *and others.*)

```
⟨/luatex⟩
```

The module declaration

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

local ProvidesLuaModule = {
    name          = "tagpdf",
    version       = "0.99s",       --TAGVERSION
    date          = "2025-06-27", --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.role   will contain data related to parent-child relations.
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/logging functions.
local functions 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,index): outputs the object reference for the object name
 ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of po
 ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log leve
 ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc's if the current l
 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

 ltx.tag.get_structnum: number, shows the current structure number
 ltx.tag.get_structnum_next: number, shows the next structure number
--]]

```

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 structnum attribute stores the structure number. 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. The interwordspaceOff attr allows to locally suppress the insertion of real space chars, e.g. when they are inserted by other means (e.g. with `\char`).

```
local mctypeattributeid  = luatexbase.new_attribute ("g__tag_mc_type_attr")
local mccntattributeid   = luatexbase.new_attribute ("g__tag_mc_cnt_attr")
local structnumattributeid   = luatexbase.new_attribute ("g__tag_structnum_attr")
local iwspaceOffattributeid = luatexbase.new_attribute ("g__tag_interwordspaceOff_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")
```

with this token we can query the state of the softhyphen boolean and so detect if hyphens from hyphenation should be replaced by soft-hyphens.

```
local softhyphenbool = token.create("g__tag_softhyphen_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 fonthashes      = fonts.hashes
local identifiers     = fonthashes.identifiers
local fontid          = font.id

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")

local NEXT = next
local explicit_disc = 1
local regular_disc = 3
```

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

```
ltx             = ltx        or { }
ltx.tag         = ltx.tag       or { } -- user commands
ltx.__tag          = ltx.__tag        or { }
ltx.__tag.mc       = ltx.__tag.mc     or  { } -- mc data
ltx.__tag.role     = ltx.__tag.role   or  { } -- parent-child data
ltx.__tag.role.states = ltx.__tag.role.states   or  { } -- the states
ltx.__tag.role.index  = ltx.__tag.role.index    or  { } -- standard types to index
                                                  --- numbers
ltx.__tag.role.matrix = ltx.__tag.role.matrix   or  { } -- implements the matrix
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 tab
                                        -- used for pure lua tables too now!
ltx.__tag.page     = ltx.__tag.page   or  { } -- page data, currently only i->{0->mcnum,1->mc
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 User commands to access data

Code like the one in luamml will have to access the current state in some places.

\

```
local __tag_get_struct_num =
 function()
  local a = token.get_macro("g__tag_struct_stack_current_tl")
  return a
 end

local __tag_get_struct_counter =
 function()
  local a = tex.getcount("c@g__tag_struct_abs_int")
  return a
 end

local __tag_get_struct_num_next =
 function()
  local a = tex.getcount("c@g__tag_struct_abs_int") + 1
  return a
 end

ltx.tag.get_struct_num = __tag_get_struct_num
ltx.tag.get_struct_counter = __tag_get_struct_counter
ltx.tag.get_struct_num_next = __tag_get_struct_num_next
```

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

3 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 of 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 of 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 of 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 of 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 of 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 of definition for* `ltx.__tag.trace.show_struct_data`.)

4 Helper functions

4.1 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 of 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 of definition for* `__tag_get_mathsubtype`.)

ltx.__tag.tables.role_tag_attribute The first is a table with key a tag and value a number (the attribute) The second is an array with the attribute value as key.

```
ltx.__tag.tables.role_tag_attribute = {}
ltx.__tag.tables.role_attribute_tag = {}
```

(*End of definition for* `ltx.__tag.tables.role_tag_attribute`.)

ltx.__tag.func.alloctag

```
local __tag_alloctag =
 function (tag)
   if not ltx.__tag.tables.role_tag_attribute[tag] then
    table.insert(ltx.__tag.tables.role_attribute_tag,tag)
    ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
    __tag_log  ("Add "..tag.." "..ltx.__tag.tables.role_tag_attribute[tag],3)
   end
 end
ltx.__tag.func.alloctag = __tag_alloctag
```

(*End of definition for* `ltx.__tag.func.alloctag`.)

`__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.role_tag_attribute[tag] then
    a= ltx.__tag.tables.role_tag_attribute[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 of 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 string for lua, while the `output` function outputs to tex.

```
local __tag_get_tag_from =
 function  (num)
  if ltx.__tag.tables.role_attribute_tag[num] then
   a = ltx.__tag.tables.role_attribute_tag[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 of 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 of 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 of 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)
 __tag_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 of 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
 __tag_log ("INFO MC-KID-NUMBERS: " .. mcnum .. "has " .. num .. "KIDS",4)
 return num
end
```

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

4.2 Functions to insert the pdf literals

`__tag_backend_create_emc_node`
`__tag_insert_emc_node`

This insert the emc node. We support also dvips and dvipdfmx backend

```
local __tag_backend_create_emc_node
if tex.outputmode == 0 then
 if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
  function __tag_backend_create_emc_node ()
    local emcnode = nodenew("whatsit","special")
      emcnode.data = "pdf:code EMC"
    return emcnode
  end
 else -- assume a dvips variant
  function __tag_backend_create_emc_node ()
    local emcnode = nodenew("whatsit","special")
      emcnode.data = "ps:SDict begin mark /EMC pdfmark end"
    return emcnode
  end
 end
else -- pdf mode
  function __tag_backend_create_emc_node ()
    local emcnode = nodenew("whatsit","pdf_literal")
      emcnode.data = "EMC"
      emcnode.mode=1
    return emcnode
  end
end

local function __tag_insert_emc_node (head,current)
  local emcnode= __tag_backend_create_emc_node()
  head = node.insert_before(head,current,emcnode)
  return head
end
```

(*End of definition for* `__tag_backend_create_emc_node` *and* `__tag_insert_emc_node`.)

`__tag_backend_create_bmc_node`
`__tag_insert_bmc_node`

This inserts a simple bmc node

```
local __tag_backend_create_bmc_node
if tex.outputmode == 0 then
 if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
  function __tag_backend_create_bmc_node (tag)
    local bmcnode = nodenew("whatsit","special")
    bmcnode.data = "pdf:code /"..tag.." BMC"
    return bmcnode
  end
 else -- assume a dvips variant
  function __tag_backend_create_bmc_node (tag)
    local bmcnode = nodenew("whatsit","special")
    bmcnode.data = "ps:SDict begin mark/"..tag.." /BMC pdfmark end"
    return bmcnode
  end
 end
else -- pdf mode
  function __tag_backend_create_bmc_node (tag)
    local bmcnode = nodenew("whatsit","pdf_literal")
    bmcnode.data = "/"..tag.." BMC"
    bmcnode.mode=1
    return bmcnode
  end
end

local function __tag_insert_bmc_node (head,current,tag)
 local bmcnode = __tag_backend_create_bmc_node (tag)
 head = node.insert_before(head,current,bmcnode)
 return head
end
```

(*End of definition for* `__tag_backend_create_bmc_node` *and* `__tag_insert_bmc_node`.)

`__tag_backend_create_bdc_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 __tag_backend_create_bdc_node

if tex.outputmode == 0 then
 if token.get_macro("c_sys_backend_str") == "dvipdfmx" then
  function __tag_backend_create_bdc_node (tag,dict)
    local bdcnode = nodenew("whatsit","special")
    bdcnode.data = "pdf:code /"..tag.."<<"..dict..">> BDC"
    return bdcnode
  end
 else -- assume a dvips variant
  function __tag_backend_create_bdc_node (tag,dict)
    local bdcnode = nodenew("whatsit","special")
    bdcnode.data = "ps:SDict begin mark/"..tag.."<<"..dict..">> /BDC pdfmark end"
    return bdcnode
  end
 end
else -- pdf mode
  function __tag_backend_create_bdc_node (tag,dict)
    local bdcnode = nodenew("whatsit","pdf_literal")
    bdcnode.data = "/"..tag.."<<"..dict..">> BDC"
    bdcnode.mode=1
    return bdcnode
  end
end

local function __tag_insert_bdc_node (head,current,tag,dict)
 bdcnode= __tag_backend_create_bdc_node (tag,dict)
 head = node.insert_before(head,current,bdcnode)
 return head
end
```

(*End of definition for* `__tag_backend_create_bdc_node` *and* `__tag_insert_bdc_node`.)

`__tag_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.

```
local function __tag_pdf_object_ref (name,index)
   local object
   if ltx.pdf.object_id then
     object = ltx.pdf.object_id (name,index) ..' 0 R'
   else
     local tokenname = 'c__pdf_object_'..name..'/'..index..'_int'
     object = token.create(tokenname).mode ..' 0 R'
   end
   return object
end
ltx.__tag.func.pdf_object_ref = __tag_pdf_object_ref
```

(*End of definition for* `__tag_pdf_object_ref`.)

5 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
 if tex.outputmode == 0 then
  -- ignore dvi mode for now
 else
  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
```

(*End of 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 of 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
      default_currfontid = glyph.font
      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
      -- __tag_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)
        --  changed 2024-01-18, issue #72
        nodesetattribute(glyph.next,iwfontattributeid,default_currfontid)
      -- 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 of definition for* `__tag_mark_spaces`.)

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

These 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 of definition for* `__tag_activate_mark_space`, `ltx.__tag.func.markspaceon`, *and* `ltx.__tag.func.markspaceoff`.)

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

```
local default_space_char = nodenew(GLYPH)
local default_fontid     = fontid("TU/lmr/m/n/10")
local default_currfontid = fontid("TU/lmr/m/n/10")
default_space_char.char  = 32
default_space_char.font  = default_fontid
```

And a function to check as best as possible if a font has a space:

```
local function __tag_font_has_space (fontid)
 t= fonts.hashes.identifiers[fontid]
 if luaotfload.aux.slot_of_name(fontid,"space")
    or t.characters and t.characters[32] and t.characters[32]["unicode"]==32
 then
    return true
 else
    return false
 end
end
```

`__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 = -1
      if not nodehasattribute(n,iwspaceOffattributeid) then
        spaceattr = nodegetattribute(n,iwspaceattributeid)  or -1
      end
      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)
          __tag_log ("INFO SPACE-FUNCTION-FONT: ".. tostring(curfont),3)
          if curfont and
            -- luaotfload.aux.slot_of_name(curfont,"space")
            __tag_font_has_space (curfont)
          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
    box.head = head
  end
end

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

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

6 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
  __tag_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(catlatex,"[")
    end
    for i,kidstable in ipairs( ltx.__tag.mc[mcnum]["kids"] ) do
     local kidnum  = kidstable["kid"]
     local kidpage = kidstable["page"]
     local kidpageobjnum = pdfpageref(kidpage)
     __tag_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(catlatex,"]")
    end
   else
    -- this is typically not a problem, e.g. empty hbox in footer/header can
    -- trigger this warning.
    __tag_log("WARN TEX-MC-INSERT-NO-KIDS: "..mcnum.." has no kids",2)
    if single==1 then
```

```
      tex.sprint(catlatex,"null")
    end
   end
  else
   __tag_log("WARN TEX-MC-INSERT-MISSING: "..mcnum.." doesn't exist",0)
  end
end
```

(*End of 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)
 __tag_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 of 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
 __tag_log("INFO TAG-MC-INTO-PAGE: page " .. page ..
                   ": inserting MCID " .. mcpagecnt .. " => " .. mcnum,3)
end
```

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

ltx.__tag.func.update_mc_attributes

This updates the mc-attributes of a box. It should only be used on boxes which don't contain structure elements. The arguments are a box, the mc-num and the type (as a number)

```
local function __tag_update_mc_attributes (head,mcnum,type)
 for n in node.traverse(head) do
   node.set_attribute(n,mccntattributeid,mcnum)
   node.set_attribute(n,mctypeattributeid,type)
   if n.id == HLIST or n.id == VLIST then
     __tag_update_mc_attributes (n.list,mcnum,type)
   end
 end
 return head
end
ltx.__tag.func.update_mc_attributes = __tag_update_mc_attributes
```

(*End of definition for* `ltx.__tag.func.update_mc_attributes`.)

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. (???)
  __tag_log ("INFO TAG-ABSPAGE: " .. abspage,3)
  __tag_log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
                  " prev "..mccntprev ..
                  " type prev "..mctypeprev,4)
  __tag_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)
    __tag_log ("INFO TAG-HEAD: " ..
                     node.type(node.getid(head))..
                     " MC"..tostring(mccnthead)..
                     " => TAG " .. tostring(mctypehead)..
                     " => ".. tostring(taghead),3)
  else
    __tag_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
    __tag_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",mctype
    elseif n.id == VLIST then -- enter the vlist
     mcopen,mcpagecnt,mccntprev,mctypeprev=
      ltx.__tag.func.mark_page_elements (n,mcpagecnt,mccntprev,mcopen,"INTERNAL VLIST",mctype
    elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but t
                                   -- 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
     __tag_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
      __tag_log("INFO TAG-MATH-SUBTYPE: "..
        node.type(node.getid(n)).." "..__tag_get_mathsubtype(n),4)
     end
     -- endmath
     __tag_log("INFO TAG-MC-COMPARE: current "..
              mccnt.." prev "..mccntprev,4)
     if mccnt~=mccntprev then -- a new mc chunk
      __tag_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
       __tag_log ("INFO TAG-INSERT-EMC: " ..
         mcpagecnt .. " MCOPEN = " .. mcopen,3)
       if mcopen ~=0 then
        __tag_log ("WARN TAG-OPEN-MC: " .. mcopen,1)
       end
      end
      if ltx.__tag.mc[mccnt] then
       if ltx.__tag.mc[mccnt]["artifact"] then
        __tag_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
        __tag_log("INFO TAG-INSERT-TAG: "..
                        tostring(tag),3)
        mcpagecnt = mcpagecnt +1
        __tag_log ("INFO TAG-INSERT-BDC: "..mcpagecnt,3)
        local dict= "/MCID "..mcpagecnt
        if ltx.__tag.mc[mccnt]["raw"] then
         __tag_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
         __tag_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]["lang"] then
         __tag_log("INFO TAG-USE-LANG: "..
            tostring(ltx.__tag.mc[mccnt]["lang"]),3)
         dict= dict .. " " .. ltx.__tag.mc[mccnt]["lang"]
        end
        if ltx.__tag.mc[mccnt]["actualtext"] then
         __tag_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
        __tag_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
        __tag_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)
    __tag_log ("INFO TAG-ENDHEAD: " ..
                       node.type(node.getid(head))..
                      " MC"..tostring(mccnthead)..
                      " => TAG "..tostring(mctypehead)..
                      " => "..tostring(taghead),4)
  else
    __tag_log ("INFO TAG-ENDHEAD: ".. tostring(head),4)
  end
  __tag_log ("INFO TAG-QUITTING-BOX "..
                     tostring(name)..
                    " TYPE ".. node.type(node.getid(box)),4)
  return mcopen,mcpagecnt,mccntprev,mctypeprev
 end

```

(*End of 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 = __tag_backend_create_emc_node ()
  local list = box.list
  if list then
     list = node.insert_after (list,node.tail(list),emcnode)
     mcopen = mcopen - 1
     __tag_log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
  else
     __tag_log ("WARN SHIPOUT-UPS: this shouldn't happen",0)
  end
  if mcopen ~=0 then
     __tag_log ("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
  end
 end
end
```

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

7 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]
     __tag_log("INFO PARENTTREE-NUM:  page "..
                    page.." has "..mcchunks.."+1 Elements ",4)
     for i=0,mcchunks do
     -- what does this log??
      __tag_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)
      __tag_log("INFO PARENTTREE-STRUCT-OBJREF:  =====>"..
        tostring(objref),5)
      numsentry = pdfpage .. " [".. objref .. "]"
      __tag_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 .. "] "
      __tag_log("INFO PARENTTREE-NUMENTRY: page " ..
        page.. " num entry = ".. numsentry,3)
     end
    else
      __tag_log ("INFO PARENTTREE-NO-DATA: page "..page,3)
      numsentry = pdfpage.." []"
    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 of definition for* `ltx.__tag.func.fill_parent_tree_line` *and* `ltx.__tag.func.output_parenttree`.)

s_softhyphen_pre process_softhyphen_post

First some local definitions. Since these are only needed locally everything gets wrapped into a block.

```
do
  local properties = node.get_properties_table()
  local is_soft_hyphen_prop = 'tagpdf.rewrite-softhyphen.is_soft_hyphen'
  local hyphen_char = 0x2D
  local soft_hyphen_char = 0xAD
```

A lookup table to test if the font supports the soft hyphen glyph.

```
  local softhyphen_fonts = setmetatable({}, {__index = function(t, fid)
    local fdir = identifiers[fid]
    local format = fdir and fdir.format
    local result = (format == 'opentype' or format == 'truetype')
    local characters = fdir and fdir.characters
    result = result and (characters and characters[soft_hyphen_char]) ~= nil
    t[fid] = result
    return result
  end})
```

A pre shaping callback to mark hyphens as being hyphenation hyphens. This runs before shaping to avoid affecting hyphens moved into discretionaries during shaping.

```
  local function process_softhyphen_pre(head, _context, _dir)
    if softhyphenbool.mode ~= truebool.mode then return true end
    for disc, sub in node.traverse_id(DISC, head) do
      if sub == explicit_disc or sub == regular_disc then
```

```
        for n, _ch, _f in node.traverse_char(disc.pre) do
          local props = properties[n]
          if not props then
            props = {}
            properties[n] = props
          end
          props[is_soft_hyphen_prop] = true
        end
      end
    end
    return true
  end

```

Finally do the actual replacement after shaping. No checking for double processing here since the operation is idempotent.

```
  local function process_softhyphen_post(head, _context, _dir)
    if softhyphenbool.mode ~= truebool.mode then return true end
    for disc, sub in node.traverse_id(DISC, head) do
      for n, ch, fid in node.traverse_glyph(disc.pre) do
        local props = properties[n]
        if softhyphen_fonts[fid] and ch == hyphen_char and props and props[is_soft_hyphen_pro
          n.char = soft_hyphen_char
          props.glyph_info = nil
        end
      end
    end
    return true
  end

  luatexbase.add_to_callback('pre_shaping_filter', process_softhyphen_pre, 'tagpdf.rewrite-
softhyphen')
  luatexbase.add_to_callback('post_shaping_filter', process_softhyphen_post, 'tagpdf.rewrite-
softhyphen')
end
```

(*End of definition for* `process_softhyphen_pre` `process_softhyphen_post`. *This function is documented on page* **??**.)

8 parent-child rules

`role_get_parent_child_rule`
`ltx.__tag.func.role_get_parent_child_rule`

```
local function role_get_parent_child_rule (parent,child)
   local state=
   ltx.__tag.role.matrix[ltx.__tag.role.index[parent]]
   and ltx.__tag.role.matrix[ltx.__tag.role.index[parent]][ltx.__tag.role.index[child]] or 0
   return state
end
ltx.__tag.func.role_get_parent_child_rule=role_get_parent_child_rule
```

(*End of definition for* `role_get_parent_child_rule` *and* `ltx.__tag.func.role_get_parent_child_rule`. *This function is documented on page* **??**.)

`check_update_stashed`
`check_parent_child_rules`
`ltx.__tag.func.check_parent_child_rules`

These function allows to check the parent-child rules for the current set of structures. It should normally be used at the end of the document. Some stashed structures can still have a parentrole setting containing the STASHED keyword, there must be updated first, this is done with a helper command. To avoid that a faulty structure (where e.g. two structures point to each other) creates an endless loop we check for the real parent only for 10 loops.

```
function check_update_stashed (struct,loglevel,loop)
 loop = (loop or 0)  + 1
 if loop > 10 then
  __tag_log ('Warning: Too deeply nested stashed structures',0)
  return
 end
 __tag_log ('updating parentrole for stashed structure '..struct,loglevel)
 local parent = ltx.__tag.tables['g__tag_struct_'..struct..'_prop']['parentnum']
 if parent then
   local ptag =
    string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole'], "{(.-)}{(.-)}")
   if ptag == 'STASHED' then
   -- look at the parent and update it first
     check_update_stashed (parent,loglevel,loop)
   end
   -- now copy the parent role from the parent
     ltx.__tag.tables['g__tag_struct_'..struct..'_prop']['parentrole']
        =
     ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole']
   __tag_log
    ('new parentrole: ' .. ltx.__tag.tables['g__tag_struct_'..struct..'_prop']['parentrole'],
 else
  __tag_log ('Warning: structure '..struct.. 'has no parent.',0)
 end
end

function check_parent_child_rules (loglevel)
 texio.write_nl('\n')
 __tag_log ('checking parent-child rules ...' ,0)
 for i=2,ltx.tag.get_struct_counter() do
  local t,tNS=
    string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['tag'], "{(.-)}{(.-)}")
  local r,rNS=
    string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['rolemap'], "{(.-)}{(.-)}")
  local p,pNS=
    string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['parentrole'], "{(.-)}{(.-)}")
  local parent=ltx.__tag.tables['g__tag_struct_'..i..'_prop']['parentnum']
  if parent then
    __tag_log (i..': '.. t..':'..tNS,loglevel)
    __tag_log (i..': '.. r..':'..rNS,loglevel)
    __tag_log (i..': '.. p..':'..pNS,loglevel)
    __tag_log ('parent of ' ..i..': '.. parent,loglevel )
    if p == 'STASHED' then
```

```
     check_update_stashed  (i,loglevel,0)
     p,pNS=
       string.match(ltx.__tag.tables['g__tag_struct_'..i..'_prop']['parentrole'], "{(.-
)}{(.-)}")
    end
    local pt,ptNS=
    string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['tag'], "{(.-
)}{(.-)}")
    local pr,prNS=
    string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['rolemap'], "{(.-
)}{(.-)}")
    local pp,ppNS=
    string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole'], "{(.-
)}{(.-)}")
    if pp == 'STASHED' then
     check_update_stashed  (parent,loglevel,0)
     pp,ppNS=
      string.match(ltx.__tag.tables['g__tag_struct_'..parent..'_prop']['parentrole'], "{(.-
)}{(.-)}")
    end
    __tag_log (parent..': '.. pt..':'..ptNS,loglevel)
    __tag_log (parent..': '.. pr..':'..prNS,loglevel)
    __tag_log (parent..': '.. pp..':'..ppNS,loglevel)
   -- now check the rule.
   -- at first rolemap of child against rolemap of parent.
    local state=ltx.__tag.func.role_get_parent_child_rule (pr,r)
    __tag_log ('rule of '..pr.."->"..r..' is '..state,loglevel)
    -- if the state is 7 we check against parentrole of the parent
    if state == 7 then
     state=ltx.__tag.func.role_get_parent_child_rule (pp,r)
     __tag_log ('Parent-Child relation '..pp.."->"..r..' is '..state,loglevel)
    end
    if state == 0 then
      __tag_log
      ('Warning: Parent-Child relation '
       ..ptNS..':'..pt..' -> '..tNS..':'..t..' is unknown',0)
      __tag_log
       ('Structure ' ..parent..' -> '..i,0)
    end
    if state == -1 then
     __tag_log
      ('Warning: Parent-Child relation '
       ..ptNS..':'..pt..' -> '..tNS..':'..t..' is not allowed',0)
     __tag_log
       ('Structure ' ..parent..' -> '..i,0)
    end
    -- check also for MC
    state =ltx.__tag.func.role_get_parent_child_rule ( r ,'MC')
    local curtag=r
    if state == 7 then
     state =ltx.__tag.func.role_get_parent_child_rule ( p ,'MC')
     local curtag=p
    end
    if state == -1 then
```

```
     if ltx.__tag.struct[i] and NEXT(ltx.__tag.struct[i]) then
       __tag_log
       ('Warning: Real content (MC) is not allowed in ' ..curtag,0)
     end
    end
    __tag_log('=====================================',loglevel)
   end
  end -- end for
 end

ltx.__tag.func.check_parent_child_rules=check_parent_child_rules

```

(*End of definition for* `check_update_stashed`*,* `check_parent_child_rules`*, and* `ltx.__tag.func.check_-parent_child_rules`*. These functions are documented on page* **??**.)

9 Link annotations

If the linksplit code has been loaded we use it to add the OBJR of links to the structure tree.

```
  if luatexbase.callbacktypes['linksplit'] then
   luatexbase.add_to_callback('linksplit', function(start_link, position)
     local structnum =
       node.get_attribute(start_link,luatexbase.attributes.g__tag_structnum_attr)
     if structnum and structnum > -1 then
      local s = ltx.__tag.tables['g__tag_struct_'..structnum..'_prop']['rolemap']
      if s and (string.find(s,'Link') or string.find(s,'Reference')) then
           local struct_insert_annot_shipout = token.create'__tag_struct_insert_annot_shipout
           local parentnum = tex.count['c@g__tag_parenttree_obj_int']
           start_link.link_attr =
              start_link.link_attr ..
              ' /LTEX_position /' .. position ..
              '/StructParent ' .. parentnum
           tex.sprint(catlatex,struct_insert_annot_shipout,'{'..
              structnum..'}{'..
              start_link.objnum..' 0 R}{'..
              parentnum ..'}')
           -- the counter must be set explicitly as struct_insert_annot_shipout doesn't do it
           tex.setcount('global','c@g__tag_parenttree_obj_int',parentnum +1)
            __tag_log(position .. " link part has object id " .. start_link.objnum .. " and s
       else
        __tag_log('Warning: Link not in Link or Reference structure element',0)
        __tag_log('OBJR not created',0)
        __tag_log('',0)
       end
     end
   end, 'tagpdf')
  end
```

⟨/lua⟩

Part X
The **tagpdf-roles** module Tags, roles and namespace code Part of the tagpdf package

```
add-new-tag (setup-key)
tag (rolemap-key)
namespace (rolemap-key)
role (rolemap-key)
role-namespace (rolemap-key)
```

The `add-new-tag` key can be used in `\tagpdfsetup` to declare and rolemap new tags. It takes as value a key-value list or a simple `new-tag/old-tag`.

The key-value list knows the following keys:

`tag` This is the name of the new tag as it should then be used in `\tagstructbegin`.

`namespace` This is the namespace of the new tag. The value should be a shorthand of a namespace. The allowed values are currently `pdf`, `pdf2`, `mathml`,`latex`, `latex-book` and `user`. The default value (and recommended value for a new tag) is `user`. The public name of the user namespace is `tag/NS/user`. This can be used to reference the namespace e.g. in attributes.

`role` This is the tag the tag should be mapped too. In a PDF 1.7 or earlier this is normally a tag from the `pdf` set, in PDF 2.0 from the `pdf`, `pdf2` and `mathml` set. It can also be a user tag. The tag must be declared before, as the code retrieves the class of the new tag from it. The PDF format allows mapping to be done transitively. But tagpdf can't/won't check such unusual role mapping.

`role-namespace` If the role is a known tag the default value is the default namespace of this tag. With this key a specific namespace can be forced.

Namespaces are mostly a PDF 2.0 property, but it doesn't harm to set them also in a PDF 1.7 or earlier.

`\tag_check_child:nnTF` `\tag_check_child:nnTF {⟨tag⟩} {⟨namespace⟩} {⟨true code⟩} {⟨false code⟩}`

This checks if the tag ⟨*tag*⟩ from the name space ⟨*namespace*⟩ can be used at the current position. In tagpdf-base it is always true.

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-roles-code} {2025-06-27} {0.99s}
 {part of tagpdf - code related to roles and structure names}
⟨/header⟩
```

1 Code related to roles and structure names

```
⟨*package⟩
```

1.1 Variables

Tags are used in structures (`\tagstructbegin`) and mc-chunks (`\tagmcbegin`).

They have a name (a string), in lua a number (for the lua attribute), and in PDF 2.0 belong to one or more name spaces, with one being the default name space.

Tags of structures are classified, e.g. as grouping, inline or block level structure (and a few special classes like lists and tables), and must follow containments rules depending on their classification (for example a inline structure can not contain a block level structure). New tags inherit their classification from their rolemapping to the standard namespaces (`pdf` and/or `pdf2`). We store this classification as it will probably be needed for tests but currently the data is not much used. The classification for math (and the containment rules) is unclear currently and so not set.

The attribute number is only relevant in lua and only for the MC chunks (so tags with the same name from different names spaces can have the same number), and so only stored if luatex is detected.

Due to the namespaces the storing and processing of tags and there data are different in various places for PDF 2.0 and PDF <2.0, which makes things a bit difficult and leads to some duplications. Perhaps at some time there should be a clear split.

This are the main variables used by the code:

`\g__tag_role_tags_NS_prop` This is the core list of tag names. It uses tags as keys and the shorthand (e.g. pdf2, or mathml) of the default name space as value.

In pdf 2.0 the value is needed in the structure dictionaries.

`\g__tag_role_tags_class_prop` This contains for each tag a classification type. It is used in pdf <2.0.

`\g__tag_role_NS_prop` This contains the names spaces. The values are the object references. They are used in pdf 2.0.

`\g__tag_role_rolemap_prop` This contains for each tag the role to a standard tag. It is used in pdf<2.0 for tag checking and to fill at the end the RoleMap dictionary.

`g_@@_role/RoleMap_dict` This dictionary contains the standard rolemaps. It is relevant only for pdf <2.0.

`\g__tag_role_NS_<ns>_prop` This prop contains the tags of a name space and their role. The props are also use for remapping. As value they contain two brace groups: tag and namespace. In pdf <2.0 the namespace is empty.

`\g__tag_role_NS_<ns>_class_prop` This prop contains the tags of a name space and their type. The value is only needed for pdf 2.0.

`\g__tag_role_index_prop` This prop contains the standard tags (pdf in pdf<2.0, pdf,pdf2 + mathml in pdf 2.0) as keys, the values are a two-digit number. These numbers are used to get the containment rule of two tags from the intarray.

`\g__tag_role_tags_NS_prop` This is the core list of tag names. It uses tags as keys and the shorthand (e.g. pdf2, or mathml) of the default name space as value. We store the default name space also in pdf <2.0, even if not needed: it doesn't harm and simplifies the code. 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
```

(*End of definition for* `\g__tag_role_tags_NS_prop`.)

`\g__tag_role_tags_class_prop` With pdf 2.0 we store the class in the NS dependent props. With pdf <2.0 we store for now the type(s) of a tag in a common prop. Tags that are rolemapped should get the type from the target.

```
\prop_new:N     \g__tag_role_tags_class_prop
```

(*End of definition for* `\g__tag_role_tags_class_prop`.)

`\g__tag_role_NS_prop` This holds the list of supported name spaces. The keys are the name tagpdf will use, the values the object reference. The urls identifier are stored in related dict 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)

latex https://www.latex-project.org/ns/dflt

latex-book https://www.latex-project.org/ns/book

More namespaces are possible and their objects references and their rolemaps 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 the object reference as it will be needed rather often.

```
\prop_new:N \g__tag_role_NS_prop
```

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

`\g__tag_role_index_prop` This prop contains the standard tags (pdf in pdf<2.0, pdf,pdf2 + mathml in pdf 2.0) as keys, the values are a two-digit number. These numbers are used to get the containment rule of two tags from the intarray.

```
\prop_new:N \g__tag_role_index_prop
```

(*End of definition for* `\g__tag_role_index_prop`.)

`\l__tag_role_debug_prop` This variable is used to pass more infos to debug messages.

```
\prop_new:N \l__tag_role_debug_prop
```

(*End of definition for* `\l__tag_role_debug_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_tag_namespace_tmpb_tl` %
`\l__tag_role_role_tmpa_tl`
`\l__tag_role_role_namespace_tmpa_tl`
`\l__tag_role_tmpa_seq`

```
\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_tag_namespace_tmpb_tl
\tl_new:N \l__tag_role_role_tmpa_tl
\tl_new:N \l__tag_role_role_namespace_tmpa_tl
\seq_new:N\l__tag_role_tmpa_seq
```

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

1.2 Namespaces

The following commands setups a name space. With pdf version <2.0 this is only a prop with the rolemap. With pdf 2.0 a dictionary must be set up. Such a name space 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 dict to collect the rolemaps if needed, and a property with the tags of the name space and their rolemapping for loops. It is unclear if a reference to a schema file will be ever needed, but it doesn't harm ….

`g__tag_role/RoleMap_dict`
`\g__tag_role_rolemap_prop`

This is the object which contains the normal RoleMap. It is probably not needed in pdf 2.0 but currently kept.

```
\pdfdict_new:n {g__tag_role/RoleMap_dict}
\__tag_prop_new:N \g__tag_role_rolemap_prop
```

(*End of definition for* `g__tag_role/RoleMap_dict` *and* `\g__tag_role_rolemap_prop`.)

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

`\__tag_role_NS_new:nnn`

```
\pdf_version_compare:NnTF < {2.0}
 {
   \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
    {
      \__tag_prop_new:c { g__tag_role_NS_#1_prop }
      \prop_new:c { g__tag_role_NS_#1_class_prop }
      \prop_gput:Nne \g__tag_role_NS_prop {#1}{}
    }
 }
 {
  \cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3
    {
      \__tag_prop_new:c { g__tag_role_NS_#1_prop }
      \prop_new:c { g__tag_role_NS_#1_class_prop }
      \pdf_object_new:n {tag/NS/#1}
      \pdfdict_new:n     {g__tag_role/Namespace_#1_dict}
      \pdf_object_new:n {__tag/RoleMapNS/#1}
      \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__tag_tmpa_str
      \tl_if_empty:NF \l__tag_tmpa_str
        {
          \pdfdict_gput:nne
            {g__tag_role/Namespace_#1_dict}
            {NS}
            {\l__tag_tmpa_str}
        }
      %RoleMapNS is added in tree
      \tl_if_empty:nF  {#3}
       {
         \pdfdict_gput:nne{g__tag_role/Namespace_#1_dict}
          {Schema}{#3}
       }
      \prop_gput:Nne \g__tag_role_NS_prop {#1}{\pdf_object_ref:n{tag/NS/#1}~}
    }
 }
```

(*End of 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 do not try to be really exact as it doesn't matter ...

`\c__tag_role_userNS_id_str`

```
\str_const:Ne \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 of definition for* `\c__tag_role_userNS_id_str`.)

Now we setup the standard names spaces. The mathml space is loaded also for pdf < 2.0 but not added to RoleMap unless a boolean is set to true with `tagpdf-setup{mathml-tags}`.

```
\bool_new:N \g__tag_role_add_mathml_bool
\__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}{}
\__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dflt}{}
\__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book}{}
\exp_args:Nne
  \__tag_role_NS_new:nnn {user}{\c__tag_role_userNS_id_str}{}
```

1.3 Adding a new tag

Both when reading the files and when setting up a tag manually we have to store data in various places.

`\__tag_role_alloctag:nnn`

This command allocates a new tag without role mapping. In the lua backend it will also record the attribute value.

```
\pdf_version_compare:NnTF < {2.0}
  {
   \sys_if_engine_luatex:TF
    {
      \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
       {
         \lua_now:e { ltx.__tag.func.alloctag ('#1') }
         \prop_gput:Nnn \g__tag_role_tags_NS_prop   {#1}{#2}
         \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
         \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
         \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
       }
    }
    {
      \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
       {
         \prop_gput:Nnn \g__tag_role_tags_NS_prop   {#1}{#2}
         \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
         \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
         \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{--UNUSED--}
       }
    }
  }
  {
   \sys_if_engine_luatex:TF
    {
      \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
       {
         \lua_now:e { ltx.__tag.func.alloctag ('#1') }
         \prop_gput:Nnn \g__tag_role_tags_NS_prop   {#1}{#2}
         \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
         \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
         \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
       }
    }
    {
      \cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3
       {
         \prop_gput:Nnn \g__tag_role_tags_NS_prop   {#1}{#2}
         \__tag_prop_gput:cnn {g__tag_role_NS_#2_prop}  {#1}{{}{}}
         \prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
         \prop_gput:cnn {g__tag_role_NS_#2_class_prop}  {#1}{#3}
       }
    }
  }
\cs_generate_variant:Nn  \__tag_role_alloctag:nnn {nno}
```

(*End of definition for* `\__tag_role_alloctag:nnn`.)

1.3.1 pdf 1.7 and earlier

`\__tag_role_add_tag:nn`

The pdf 1.7 version has only two arguments: new and rolemap name. The role must be an existing tag and should not be empty. We allow to change the role of an existing tag: as the rolemap is written at the end not confusion can happen.

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

checks and messages

```
    \__tag_check_add_tag_role:nn {#1}{#2}
    \prop_get:NnNF \g__tag_role_tags_NS_prop {#1}\l__tag_tmp_unused_tl
      {
        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
          {
            \msg_info:nnn { tag }{new-tag}{#1}
          }
      }
```

now the addition

```
    \prop_get:NnNF \g__tag_role_tags_class_prop {#2}\l__tag_tmpa_tl
      {
        \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
      }
    \__tag_role_alloctag:nno {#1}{user} { \l__tag_tmpa_tl }
```

We resolve rolemapping recursively so that all targets are stored as standard tags.

```
    \tl_if_empty:nF { #2 }
      {
        \prop_get:NnNTF \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
          {
            \__tag_prop_gput:Nno \g__tag_role_rolemap_prop {#1}{\l__tag_tmpa_tl}
          }
          {
            \__tag_prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
          }
      }
  }
\cs_generate_variant:Nn \__tag_role_add_tag:nn {oo,ne}
```

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

For the parent-child test we must be able to get the role. We use the same number of arguments as for the 2.0 command. If there is no role, we assume a standard tag. Note: this is quite fast and a move to lua doesn't improve speed.

`\__tag_role_get:nnNN`

```
\pdf_version_compare:NnT < {2.0}
 {
   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4 %#1 tag, #2 NS, #3 tlvar which hold the role tag
    {
      \prop_get:NnNF \g__tag_role_rolemap_prop {#1}#3
        {
          \tl_set:Nn #3 {#1}
        }
      \tl_set:Nn #4 {}
    }
   \cs_generate_variant:Nn \__tag_role_get:nnNN {ooNN}
 }

```

(*End of definition for* `\__tag_role_get:nnNN`.)

1.3.2 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
  {
    \__tag_check_add_tag_role:nnn {#1/#2}{#3}{#4}
    \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
      {
        \msg_info:nnn { tag }{new-tag}{#1}
      }
    \prop_if_exist:cTF
     { g__tag_role_NS_#4_class_prop }
     {
       \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
       \quark_if_no_value:NT \l__tag_tmpa_tl
        {
          \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
        }
     }
     { \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--} }
    \__tag_role_alloctag:nno {#1}{#2}{ \l__tag_tmpa_tl }
```

Do not remap standard tags. TODO add warning?

```
    \tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-}
     {
       \pdfdict_gput:nne {g__tag_role/RoleMapNS_#2_dict}{#1}
          {
            [
              \pdf_name_from_unicode_e:n{#3}
              \c_space_tl
              \pdf_object_ref:n {tag/NS/#4}
            ]
          }
     }
```

We resolve rolemapping recursively so that all targets are stored as standard tags for the tests.

```
    \tl_if_empty:nF { #2 }
      {
        \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
        \quark_if_no_value:NTF \l__tag_tmpa_tl
          {
            \__tag_prop_gput:cne { g__tag_role_NS_#2_prop } {#1}
              {{\tl_to_str:n{#3}}{\tl_to_str:n{#4}}}
          }
          {
            \__tag_prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
          }
      }
```

We also store into the pdf 1.7 rolemapping so that we can add that as fallback for pdf 1.7 processor

```
     \bool_if:NT \l__tag_role_update_bool
       {
         \tl_if_empty:nF { #3 }
```

```
          {
            \tl_if_eq:nnF{#1}{#3}
             {
              \prop_get:NnN \g__tag_role_rolemap_prop {#3}\l__tag_tmpa_tl
               \quark_if_no_value:NTF \l__tag_tmpa_tl
                {
                  \__tag_prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#3}}
                }
                {
                  \__tag_prop_gput:Nno \g__tag_role_rolemap_prop {#1}{\l__tag_tmpa_tl}
                }
              }
           }
       }
   }
\cs_generate_variant:Nn \__tag_role_add_tag:nnnn {oooo}
```

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

For the parent-child test we must be able to get the role. We use the same number of arguments as for the <2.0 command. Note: this is quite fast and a move to lua doesn't improve speed.

`\__tag_role_get:nnNN`

```
\pdf_version_compare:NnF < {2.0}
 {
   \cs_new:Npn \__tag_role_get:nnNN #1#2#3#4
     %#1 tag, #2 NS,
     %#3 tlvar which hold the role tag
     %#4 tlvar which hold the name of the target NS
    {
      \prop_if_exist:cTF {g__tag_role_NS_#2_prop}
       {
        \prop_get:cnNTF {g__tag_role_NS_#2_prop} {#1}\l__tag_get_tmpc_tl
          {
           \tl_set:Ne #3 {\exp_last_unbraced:No\use_i:nn  {\l__tag_get_tmpc_tl}}
           \tl_set:Ne #4 {\exp_last_unbraced:No\use_ii:nn {\l__tag_get_tmpc_tl}}
          }
          {
           \msg_warning:nnn { tag } {role-unknown-tag} { #1 }
           \tl_set:Nn #3 {#1}
           \tl_set:Nn #4 {#2}
          }
       }
       {
          \msg_warning:nnn { tag } {role-unknown-NS} { #2 }
          \tl_set:Nn #3 {#1}
          \tl_set:Nn #4 {#2}
       }
    }
   \cs_generate_variant:Nn \__tag_role_get:nnNN {ooNN}
 }
```

(*End of definition for* `\__tag_role_get:nnNN`.)

1.4 Helper command to read the data from files

In this section we setup the helper command to read namespace files.

__tag_role_read_namespace_line:nw This command will process a line in the name space file. The first argument is the name of the name space. The definition differ for pdf 2.0. as we have proper name spaces there. With pdf<2.0 special name spaces shouldn't update the default role or add to the rolemap again, they only store the values for later uses. We use a boolean here.

```
\bool_new:N\l__tag_role_update_bool
\bool_set_true:N \l__tag_role_update_bool
\pdf_version_compare:NnTF < {2.0}
 {
  \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
   % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
    {
      \tl_if_empty:nF { #2 }
       {
        \bool_if:NTF \l__tag_role_update_bool
         {
          \tl_if_empty:nTF {#5}
            {
              \prop_get:NnN \g__tag_role_tags_class_prop  {#3}\l__tag_tmpa_tl
              \quark_if_no_value:NT \l__tag_tmpa_tl
                {
                  \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
                }
            }
            {
              \tl_set:Nn \l__tag_tmpa_tl {#5}
            }
          \__tag_role_alloctag:nno {#2} {#1} { \l__tag_tmpa_tl }
          \tl_if_eq:nnF {#2}{#3}
           {
            \__tag_role_add_tag:nn {#2}{#3}
           }
          \__tag_prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
         }
         {
           \__tag_prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{}}
           \prop_gput:cnn {g__tag_role_NS_#1_class_prop}  {#2}{--UNUSED--}
         }
       }
    }
 }
 {
   \cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
    % #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
    {
      \tl_if_empty:nF {#2}
       {
        \tl_if_empty:nTF {#5}
         {
           \prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
           \quark_if_no_value:NT \l__tag_tmpa_tl
```

```
            {
              \tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}
            }
        }
        {
          \tl_set:Nn \l__tag_tmpa_tl {#5}
        }
       \__tag_role_alloctag:nno {#2} {#1} { \l__tag_tmpa_tl }
       \bool_lazy_and:nnT
          { ! \tl_if_empty_p:n {#3} }{! \str_if_eq_p:nn {#1}{pdf2}}
          {
           \__tag_role_add_tag:nnnn {#2}{#1}{#3}{#4}
          }
       \__tag_prop_gput:cnn {g__tag_role_NS_#1_prop}  {#2}{{#3}{#4}}
     }
   }
}
```

(*End of definition for* `\__tag_role_read_namespace_line:nw`.)

`\__tag_role_read_namespace:nn` This command reads a namespace file in the format tagpdf-ns-XX.def

```
\cs_new_protected:Npn \__tag_role_read_namespace:nn #1 #2 %name of namespace #2 name of file
  {
    \prop_if_exist:cF {g__tag_role_NS_#1_prop}
      { \msg_warning:nnn {tag}{namespace-unknown}{#1} }
    \file_if_exist:nTF { tagpdf-ns-#2.def }
     {
       \ior_open:Nn \g_tmpa_ior {tagpdf-ns-#2.def}
       \msg_info:nnn {tag}{read-namespace}{#2}
       \ior_map_inline:Nn \g_tmpa_ior
         {
           \__tag_role_read_namespace_line:nw {#1} ##1,,,,\q_stop
         }
       \ior_close:N\g_tmpa_ior
     }
     {
      \msg_info:nnn{tag}{namespace-missing}{#2}
     }
  }

```

(*End of definition for* `\__tag_role_read_namespace:nn`.)

`\__tag_role_read_namespace:n` This command reads the default namespace file.

```
\cs_new_protected:Npn \__tag_role_read_namespace:n #1 %name of namespace
  {
    \__tag_role_read_namespace:nn {#1}{#1}
  }
```

(*End of definition for* `\__tag_role_read_namespace:n`.)

1.5 Reading the default data

The order is important as we want pdf2 and latex as default: if two namespace define the same tag, the last one defines which one is used if the namespace is not explicitly given.

```
\__tag_role_read_namespace:n {pdf}
\__tag_role_read_namespace:n {pdf2}
\__tag_role_read_namespace:n {mathml}
```

in pdf 1.7 the following namespaces should only store the settings for later use:

```
\bool_set_false:N\l__tag_role_update_bool
\__tag_role_read_namespace:n {latex-book}
\bool_set_true:N\l__tag_role_update_bool
\__tag_role_read_namespace:n {latex}
\__tag_role_read_namespace:nn {latex} {latex-lab}
\__tag_role_read_namespace:n {pdf}
\__tag_role_read_namespace:n {pdf2}
```

But is the class provides a `\chapter` command then we switch

```
\pdf_version_compare:NnTF < {2.0}
  {
    \hook_gput_code:nnn {begindocument}{tagpdf}
      {
        \bool_lazy_and:nnT
         {
           \cs_if_exist_p:N \chapter
         }
         {
           \cs_if_exist_p:N \c@chapter
         }
         {
           \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
             {
               \__tag_role_add_tag:ne {#1}{\use_i:nn #2\c_empty_tl\c_empty_tl}
             }
         }
      }
  }
  {
    \hook_gput_code:nnn {begindocument}{tagpdf}
      {
        \bool_lazy_and:nnT
         {
           \cs_if_exist_p:N \chapter
         }
         {
           \cs_if_exist_p:N \c@chapter
         }
         {
           \prop_map_inline:cn{g__tag_role_NS_latex-book_prop}
             {
               \prop_gput:Nnn \g__tag_role_tags_NS_prop    { #1 }{ latex-book }
               \__tag_prop_gput:Nne
                \g__tag_role_rolemap_prop {#1}{\use_i:nn  #2\c_empty_tl\c_empty_tl}
             }
         }
    }
  }
```

1.6 Parent-child rules

PDF define various rules about which tag can be a child of another tag. The following code implements the matrix to allow to use it in tests.

\g__tag_role_parent_child_intarray This intarray will store the rule as a number. For parent nm and child ij (n,m,i,j digits) the rule is at position nmij. As we have around 56 tags, we need roughly a size 6000.

```
\intarray_new:Nn \g__tag_role_parent_child_intarray {6000}
```

(*End of definition for* `\g__tag_role_parent_child_intarray`.)

\c__tag_role_rules_prop
\c__tag_role_rules_num_prop These two properties map the rule strings to numbers and back. There are in tagpdf-data.dtx near the csv files for easier maintenance.

(*End of definition for* `\c__tag_role_rules_prop` *and* `\c__tag_role_rules_num_prop`.)

__tag_store_parent_child_rule:nnn The helper command is used to store the rule. It assumes that parent and child are given as 2-digit number!

```
\sys_if_engine_luatex:TF
{
  \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
    {
      \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
        {
          \intarray_gset:Nnn \g__tag_role_parent_child_intarray
           { #1#2 }{0\l__tag_tmp_unused_tl}
          \lua_now:e
           {
            ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
            ltx.__tag.role.matrix[#1][#2] = 0\l__tag_tmp_unused_tl
           }
        }
        {
          \intarray_gset:Nnn \g__tag_role_parent_child_intarray
           { #1#2 }{0}
          \lua_now:e
           {
            ltx.__tag.role.matrix[#1] = ltx.__tag.role.matrix[#1] or {}
            ltx.__tag.role.matrix[#1][#2] = 0
           }
        }
    }
}
{
  \cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child,
    {
      \prop_get:NeNTF \c__tag_role_rules_prop{#3} \l__tag_tmp_unused_tl
        {
          \intarray_gset:Nnn \g__tag_role_parent_child_intarray
           { #1#2 }{0\l__tag_tmp_unused_tl}
        }
        {
          \intarray_gset:Nnn \g__tag_role_parent_child_intarray
           { #1#2 }{0}
        }
    }
}
```

(*End of definition for* `\__tag_store_parent_child_rule:nnn`.)

1.6.1 Reading in the csv-files

This counter will be used to identify the first (non-comment) line

```
\int_zero:N  \l__tag_tmpa_int
```

Open the file depending on the PDF version

```
\pdf_version_compare:NnTF < {2.0}
  {
    \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child.csv}
  }
  {
    \ior_open:Nn \g_tmpa_ior {tagpdf-parent-child-2.csv}
  }
```

Now the main loop over the file

```
\ior_map_inline:Nn \g_tmpa_ior
  {
```

ignore lines containing only comments

```
    \tl_if_empty:nF{#1}
      {
```

count the lines ...

```
        \int_incr:N\l__tag_tmpa_int
```

put the line into a seq. Attention! empty cells are dropped.

```
        \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
        \int_compare:nNnTF {\l__tag_tmpa_int}=1
```

This handles the header line. It gives the tags 2-digit numbers.

```
          {
            \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
              {
                \prop_gput:Nne\g__tag_role_index_prop
                  {##2}
                  {\int_compare:nNnT{##1}<{10}{0}##1}
              }
          }
```

now the data lines.

```
          {
            \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 }
```

get the name of the child tag from the first column

```
            \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

get the number of the child, and store it in `\l__tag_tmpb_tl`

```
            \prop_get:NoN \g__tag_role_index_prop { \l__tag_tmpa_tl } \l__tag_tmpb_tl
```

remove column 2+3

```
            \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
            \seq_pop_left:NN\l__tag_tmpa_seq\l__tag_tmpa_tl
```

Now map over the rest. The index ##1 gives us the number of the parent, ##2 is the data.

```
            \seq_map_indexed_inline:Nn \l__tag_tmpa_seq
              {
                \exp_args:Nne
                \__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
              }
          }
      }
  }
```

close the read handle.

```
\ior_close:N\g_tmpa_ior
```

The Root, Hn and mathml tags are special and need to be added explicitly

```
\prop_get:NnN\g__tag_role_index_prop{StructTreeRoot}\l__tag_tmpa_tl
\prop_gput:Nne\g__tag_role_index_prop{Root}{\l__tag_tmpa_tl}
\prop_get:NnN\g__tag_role_index_prop{Hn}\l__tag_tmpa_tl
\pdf_version_compare:NnTF < {2.0}
  {
    \int_step_inline:nn{6}
      {
        \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
      }
  }
  {
    \int_step_inline:nn{10}
      {
        \prop_gput:Nne\g__tag_role_index_prop{H#1}{\l__tag_tmpa_tl}
      }
```

all mathml tags are currently handled identically with the exception of math and mtext

```
    \prop_get:NnN\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
    \prop_get:NnN\g__tag_role_index_prop {math}\l__tag_tmpb_tl
    \prop_get:NnN\g__tag_role_index_prop {mtext}\l__tag_tmpc_tl
    \prop_map_inline:Nn \g__tag_role_NS_mathml_prop
      {
        \prop_gput:Nno\g__tag_role_index_prop {#1} {\l__tag_tmpa_tl}
      }
    \prop_gput:Nno\g__tag_role_index_prop{math}{\l__tag_tmpb_tl}
    \prop_gput:Nno\g__tag_role_index_prop{mtext}{\l__tag_tmpc_tl}
  }
\sys_if_engine_luatex:T
 {
   \prop_map_inline:Nn\g__tag_role_index_prop
    {
      \lua_now:e { ltx.__tag.role.index['#1']=#2 }
    }
 }
```

1.6.2 Retrieving the parent-child rule

__tag_role_get_parent_child_rule:nnN This command retrieves the rule (as a number) and stores it in the tl-var. It assumes that the tags in #1 and #2 are standard tags after role mapping for which a rule exist. If the parent is one of Part, Div, NonStruct the result can be state 7, which means that a check must be repeated for the "real parent".

TODO check temporary variables. Check if the tl-var should be fix.

```
\tl_new:N \l__tag_parent_child_check_tl
\sys_if_engine_luatex:TF
 {
   \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
     % #1 parent (string, standard tag after rolemapping!)
     % #2 child (string, standard tag after rolemapping!)
     % #3 tl for state
     {
       \tl_set:Ne#3
         {
           \lua_now:e{tex.print(\int_use:N\c_document_cctab,ltx.__tag.func.role_get_parent_ch
         }
```

Debugging messages, this can perhaps go into debug mode.

```
         \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
           {
             \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
               {
                 \tl_set:Nn \l__tag_tmpa_tl {unknown}
               }
             \tl_set:Nn \l__tag_tmpb_tl {#1}
             \msg_note:nneee
               { tag }
               { role-parent-child-result }
               { #1 }
               { #2 }
               {
                 #3~(='\l__tag_tmpa_tl')
               }
           }
        \int_compare:nNnT {#3} = { 0 }
         {
          \msg_warning:nneee
            { tag }
            {role-parent-child-result}
            { #1 }
            { #2 }
            { unknown! }
         }

     }
 }
 {
   \cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
     % #1 parent (string, standard tag after rolemapping)
     % #2 child (string, standard tag after rolemapping)
     % #3 tl for state
```

```
    {
      \prop_get:NnN \g__tag_role_index_prop{#1}\l__tag_tmpa_tl
      \prop_get:NnN \g__tag_role_index_prop{#2}\l__tag_tmpb_tl
      \bool_lazy_and:nnTF
        { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
        { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
        {
```

Get the rule from the intarray

```
           \tl_set:Ne#3
             {
               \intarray_item:Nn
                \g__tag_role_parent_child_intarray
                {\l__tag_tmpa_tl\l__tag_tmpb_tl}
             }
          }
          {
            \tl_set:Nn#3 {0}
          }
```

Debugging messages, this can perhaps go into debug mode.

```
        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
          {
            \prop_get:NoNF\c__tag_role_rules_num_prop {#3} \l__tag_tmpa_tl
              {
                \tl_set:Nn \l__tag_tmpa_tl {unknown}
              }
            \tl_set:Nn \l__tag_tmpb_tl {#1}
            \msg_note:nneee
              { tag }
              { role-parent-child-result }
              { #1 }
              { #2 }
              {
                #3~(='\l__tag_tmpa_tl')
              }
          }
       \int_compare:nNnT {#3} = { 0 }
        {
         \msg_warning:nneee
           { tag }
           {role-parent-child-result}
           { #1 }
           { #2 }
           { unknown! }
        }
    }
  }
\cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnN {ooN}
```

(*End of definition for* `\__tag_role_get_parent_child_rule:nnN`*.*)

`\__tag_role_check_parent_child:nnnnN` This command rolemaps its arguments and then calls `\__tag_role_get_parent_child_rule:nnN` to retrieve the parent-child rule between both. It does not try to resolve inheritation rules of `Part`, `Div` and `NonStruct` but instead gives back the state 7. It is then the task of the caller command to find the real parent and run the check again. In pdf 2.0 the name spaces of the tags are relevant, so we have arguments for them, but in pdf <2.0 they are ignored and can be left empty.

```
\pdf_version_compare:NnTF < {2.0}
  {
   \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5
    % #1 parent tag,% not necessarly rolemapped, but often the case
    % #2 NS (empty in pdf 1.x)
    % #3 child tag, % not necessarly rolemapped, but often the case
    % #4 NS (empty in pdf 1.x)
    % #5 tl var: to give the result back.
     {
```

get the standard tags through rolemapping if needed at first the parent

```
       \prop_get:NnNTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
         {
           \tl_set:Nn \l__tag_tmpa_tl {#1}
         }
         {
           \prop_get:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
             {
               \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
             }
         }
```

now the child

```
       \prop_get:NnNTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
         {
           \tl_set:Nn \l__tag_tmpb_tl {#3}
         }
         {
           \prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
             {
               \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
             }
         }
```

if we got tags for parent and child we call the checking command

```
       \bool_lazy_and:nnTF
         { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
         { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
         {
           \__tag_role_get_parent_child_rule:ooN
             { \l__tag_tmpa_tl}
             { \l__tag_tmpb_tl}
             #5
         }
         {
           \tl_set:Nn #5 {0}
           \msg_warning:nneee
            { tag }
            {role-parent-child-result}
            { #1 }
            { #3 }
            { unknown! }
```

```
          }
      }
  }
```

and now the pdf 2.0 version

```
  {
   \cs_new_protected:Npn \__tag_role_check_parent_child:nnnnN #1 #2 #3 #4 #5 %tag,NS,tag,NS,
     {

```

If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```
        \tl_if_empty:nTF  {#2}
          {
            \tl_set:Nn \l__tag_tmpa_tl {#1}
          }
          {
            \prop_if_exist:cTF { g__tag_role_NS_#2_prop }
              {
               \prop_get:cnNTF
                  { g__tag_role_NS_#2_prop }
                  {#1}
                  \l__tag_tmpa_tl
                  {
                    \tl_set:Ne \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
                    \tl_if_empty:NT\l__tag_tmpa_tl
                      {
                        \tl_set:Nn \l__tag_tmpa_tl {#1}
                      }
                  }
                  {
                    \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
                  }
              }
              {
                \msg_warning:nnn { tag } {role-unknown-NS} { #2}
                \tl_set:Nn \l__tag_tmpa_tl {\q_no_value}
              }
          }
```

and the same for the child If the namespace is empty, we assume a standard tag, otherwise we retrieve the rolemapping from the namespace

```
        \tl_if_empty:nTF  {#4}
          {
            \tl_set:Nn \l__tag_tmpb_tl {#3}
          }
          {
            \prop_if_exist:cTF { g__tag_role_NS_#4_prop }
              {
               \prop_get:cnNTF
                 { g__tag_role_NS_#4_prop }
                 {#3}
                 \l__tag_tmpb_tl
                 {
                   \tl_set:Ne \l__tag_tmpb_tl { \tl_head:N\l__tag_tmpb_tl }
                   \tl_if_empty:NT\l__tag_tmpb_tl
                     {
                       \tl_set:Nn \l__tag_tmpb_tl {#3}
                     }
                 }
                 {
                   \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
                 }
              }
              {
                \msg_warning:nnn { tag } {role-unknown-NS} { #4}
                \tl_set:Nn \l__tag_tmpb_tl {\q_no_value}
              }
          }
```

and now get the relation

```
       \bool_lazy_and:nnTF
         { ! \quark_if_no_value_p:N \l__tag_tmpa_tl }
         { ! \quark_if_no_value_p:N \l__tag_tmpb_tl }
         {
           \__tag_role_get_parent_child_rule:ooN
             { \l__tag_tmpa_tl }
             { \l__tag_tmpb_tl }
             #5
         }
         {
           \tl_set:Nn #5 {0}
           \msg_warning:nneee
            { tag }
            {role-parent-child-result}
            { #2 : #1 }
            { #4 : #3 }
            { unknown! }
         }
     }
  }
\cs_generate_variant:Nn\__tag_role_check_parent_child:nnnnN {oonnN,ooooN}
⟨/package⟩
```

(*End of definition for* `\__tag_role_check_parent_child:nnnnN`.)

`\tag_check_child:nnTF`

```
⟨base⟩\prg_new_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF}{\prg_return_true
⟨*package⟩
\prg_set_protected_conditional:Npnn \tag_check_child:nn #1 #2 {T,F,TF} %#1 tag, #2 NS
 {
   \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
   \__tag_struct_get_role:enNN
      {\l__tag_tmpa_tl}
      {rolemap}
      \l__tag_get_parent_tmpa_tl
      \l__tag_get_parent_tmpb_tl
   \__tag_role_check_parent_child:oonnN
      { \l__tag_get_parent_tmpa_tl }
      { \l__tag_get_parent_tmpb_tl }
```

```
    {#1}{#2}
    \l__tag_parent_child_check_tl
 \int_compare:nNnT {\l__tag_parent_child_check_tl} = { \c__tag_role_rule_checkparent_tl }
    {
      \seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
      \__tag_struct_get_role:enNN
         {\l__tag_tmpa_tl}
         {parentrole}
         \l__tag_get_parent_tmpa_tl
         \l__tag_get_parent_tmpb_tl
      \__tag_role_check_parent_child:oonnN
         { \l__tag_get_parent_tmpa_tl }
         { \l__tag_get_parent_tmpb_tl }
         {#1}{#2}
         \l__tag_parent_child_check_tl
    }
 \int_compare:nNnTF {  \l__tag_parent_child_check_tl } < {0}
    {\prg_return_false:}
    {\prg_return_true:}
}
```

(*End of definition for* `\tag_check_child:nnTF`. *This function is documented on page 179.*)

1.7 Key-val user interface

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

tag (rolemap-key)
tag-namespace (rolemap-key)
role (rolemap-key)
role-namespace (rolemap-key)
role/new-tag (setup-key)
add-new-tag (deprecated)

```
\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 }
  {
     role/mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
    ,role/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__tag_tmpa_tl
       \tl_if_empty:NF \l__tag_tmpa_tl
         {
           \exp_args:NNno \seq_set_split:Nnn \l__tag_tmpa_seq { / } {\l__tag_tmpa_tl/}
           \tl_set:Ne \l__tag_role_tag_tmpa_tl  { \seq_item:Nn \l__tag_tmpa_seq {1} }
           \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l__tag_tmpa_seq {2} }
         }
      \tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
         {
           \prop_get:NoNTF
             \g__tag_role_tags_NS_prop
             { \l__tag_role_role_tmpa_tl }
             \l__tag_role_role_namespace_tmpa_tl
             {
                \prop_get:NoNF
                  \g__tag_role_NS_prop
                  { \l__tag_role_role_namespace_tmpa_tl }
                  \l__tag_tmp_unused_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:oo
             { \l__tag_role_tag_tmpa_tl }
             { \l__tag_role_role_tmpa_tl }
       }
       {
         \__tag_role_add_tag:oooo
          { \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 }
       }
    }
   ,role/map-tags .choice:
   ,role/map-tags/false .code:n = { \socket_assign_plug:nn { tag/struct/tag } {latex-tags} }
   ,role/map-tags/pdf   .code:n = { \socket_assign_plug:nn { tag/struct/tag } {pdf-tags} }
   ,role/user-NS .code:n =
     {
      \pdf_version_compare:NnF < {2.0}
       {
         \pdf_string_from_unicode:nnN{utf8/string}{https://www.latex-project.org/ns/local/#1
          \tl_if_empty:NF \l__tag_tmpa_str
            {
              \pdfdict_gput:nne
                {g__tag_role/Namespace_user_dict}
                {NS}
                {\l__tag_tmpa_str}
            }
       }
     }
```

deprecated names

```
    , mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
    , add-new-tag .meta:n = {role/new-tag={#1}}
  }
⟨/package⟩
```

(*End of definition for* `tag (rolemap-key)` *and others. These functions are documented on page 179.*)

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

`activate/space (setup-key)`
`interwordspace (deprecated)`

This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are `true`, `on`, `false`, `off`. The old name of the key `interwordspace` is still supported but deprecated.

`show-spaces (deprecated)`

This key is deprecated. Use `debug/show=spaces` instead. This key works only with luatex and shows with small red bars where spaces have been inserted. This is only for debugging and is not completely reliable (and change affect other literals and tagging), so it should be used with care.

```
⟨@@=tag⟩
⟨*header⟩
\ProvidesExplPackage {tagpdf-space-code} {2025-06-27} {0.99s}
 {part of tagpdf - code related to real space chars}
⟨/header⟩
```

1 Code for interword spaces

The code is engine/backend dependent. 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.

`activate/spaces (setup-key)`
`interwordspace (deprecated)`
`show-spaces (deprecated)`

```
⟨*package⟩
\bool_new:N\l__tag_showspaces_bool
\keys_define:nn { __tag / setup }
  {
    activate/spaces .choice:,
    activate/spaces/true .code:n =
      { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str}  },
    activate/spaces/false .code:n=
      { \msg_warning:nne {tag}{sys-no-interwordspace}{\c_sys_engine_str}  },
    activate/spaces .default:n = true,
    debug/show/spaces     .code:n = {\bool_set_true:N \l__tag_showspaces_bool},
    debug/show/spacesOff  .code:n = {\bool_set_false:N \l__tag_showspaces_bool},
```

deprecated versions:

```
    interwordspace .choices:nn = {true,on}{\keys_set:nn{__tag/setup}{activate/spaces={true}}}
    interwordspace .choices:nn = {false,off}{\keys_set:nn{__tag/setup}{activate/spaces={false
    interwordspace .default:n = {true},
    show-spaces    .choice:,
    show-spaces/true .meta:n =  {debug/show=spaces},
    show-spaces/false .meta:n = {debug/show=spacesOff},
    show-spaces .default:n = true
  }
\sys_if_engine_pdftex:T
  {
    \sys_if_output_pdf:TF
      {
        \pdfglyphtounicode{space}{0020}
        \keys_define:nn { __tag / setup }
          {
            activate/spaces/true .code:n  = { \AddToHook{shipout/firstpage}[tagpdf/space]{\pd
            activate/spaces/false .code:n = { \RemoveFromHook{shipout/firstpage}[tagpdf/space
            activate/spaces .default:n = true,
          }
      }
      {
        \keys_define:nn { __tag / setup }
          {
            activate/spaces .choices:nn = { true, false }
              { \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi}  },
            activate/spaces .default:n = true,
          }
      }
  }


\sys_if_engine_luatex:T
  {
    \keys_define:nn { __tag / setup }
      {
        activate/spaces .choice:,
        activate/spaces/true .code:n =
                                {
                                  \bool_gset_true:N \g__tag_active_space_bool
                                  \lua_now:e{ltx.__tag.func.markspaceon()}
                                },
        activate/spaces/false .code:n =
                                {
                                 \bool_gset_false:N \g__tag_active_space_bool
                                 \lua_now:e{ltx.__tag.func.markspaceoff()}
                                },
        activate/spaces .default:n = true,
        debug/show/spaces    .code:n =
                                {\lua_now:e{ltx.__tag.trace.showspaces=true}},
        debug/show/spacesOff .code:n =
                                {\lua_now:e{ltx.__tag.trace.showspaces=nil}},
      }
  }
```

(*End of definition for* `activate/spaces (setup-key)`*,* `interwordspace (deprecated)`*, and* `show-spaces (deprecated)`*. 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:
      }
  }
```

We need also a command to interrupt the insertion of real space chars in places where we want to insert manually special spaces. In pdftex this can be done with `\pdfinterwordspaceoff` and `\pdfinterwordspaceon`. These commands insert whatsits and this mean they act globally. In luatex a attribute is used to this effect, for consistency this is also set globally.

The off command sets the attributes in luatex.

`\tag_spacechar_on:`
`\tag_spacechar_off:`

```
\cs_new_protected:Npn \tag_spacechar_off: {}
\cs_new_protected:Npn \tag_spacechar_on: {}

\sys_if_engine_luatex:T
 {
   \cs_set_protected:Npn \tag_spacechar_off:
     {
       \lua_now:e
         {
           tex.setattribute
            (
             "global",
             luatexbase.attributes.g__tag_interwordspaceOff_attr,
             1
            )
         }
     }
   \cs_set_protected:Npn \tag_spacechar_on:
     {
       \lua_now:e
         {
           tex.setattribute
            (
             "global",
             luatexbase.attributes.g__tag_interwordspaceOff_attr,
             -2147483647
            )
         }
     }
 }
\sys_if_engine_pdftex:T
  {
    \sys_if_output_pdf:T
      {
       \cs_set_protected:Npn \tag_spacechar_off:
          {
            \pdfinterwordspaceoff
          }
        \cs_set_protected:Npn \tag_spacechar_on:
          {
            \pdfinterwordspaceon
          }
      }
  }
⟨/package⟩
```

(*End of definition for* `\__tag_fakespace:`*,* `\tag_spacechar_on:`*, and* `\tag_spacechar_off:`*. These functions are documented on page* **??***.*)

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.