tagpdf – A package to experiment with pdf tagging*

Ulrike Fischer†

Released 2024-02-29

Contents

1 Initialization and test if pdfmanagement is active. 7
2 base package 7
3 Package options 8
4 Packages 8
   4.1 a LastPage label 8
5 Variables 9
6 Variants of l3 commands 10
7 Label and Reference commands 11
8 Setup label attributes 11
9 Commands to fill seq and prop 12
10 General tagging commands 12
11 Keys for tagpdfsetup 14
12 loading of engine/more dependent code 15

I The tagpdf-checks module
Messages and check code
Part of the tagpdf package 17

1 Commands 17

*This file describes v0.98x, last revised 2024-02-29.
†E-mail: fischer@troubleshooting-tex.de
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Description of log messages</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>2.1 \ShowTagging command</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>2.2 Messages in checks and commands</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2.3 Messages from the ptagging code</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2.4 Warning messages from the lua-code</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2.5 Info messages from the lua-code</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2.6 Debug mode messages and code</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>2.7 Messages</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Messages</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.1 Messages related to mc-chunks</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.2 Messages related to structures</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>3.3 Attributes</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3.4 Roles</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>3.5 Miscellaneous</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>Retrieving data</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>User conditionals</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>Internal checks</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>6.1 checks for active tagging</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>6.2 Checks related to structures</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>6.3 Checks related to roles</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>6.4 Check related to mc-chunks</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>6.5 Checks related to the state of MC on a page or in a split stream</td>
<td>31</td>
</tr>
<tr>
<td>II</td>
<td>The tagpdf-user module</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Code related to \LaTeX2e user commands and document commands</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Part of the tagpdf package</td>
<td>35</td>
</tr>
<tr>
<td>1</td>
<td>Setup commands</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>Commands related to mc-chunks</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>Commands related to structures</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>Debugging</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Extension commands</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>5.1 Fake space</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>5.2 Tagging of paragraphs</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>5.3 Header and footer</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>5.4 Link tagging</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>Socket support</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>User commands and extensions of document commands</td>
<td>39</td>
</tr>
<tr>
<td>8</td>
<td>Setup and preamble commands</td>
<td>39</td>
</tr>
</tbody>
</table>
9 Commands for the mc-chunks
10 Commands for the structure
11 Socket support
12 Debugging
13 Commands to extend document commands
   13.1 Document structure
   13.2 Structure destinations
   13.3 Fake space
   13.4 Paratagging
   13.5 Header and footer
   13.6 Links
III The \texttt{tagpdf-tree} module
   Commands trees and main dictionaries
   Part of the tagpdf package
   1 Trees, pdfmanagement and finalization code
      1.1 Check structure
      1.2 Catalog: MarkInfo and StructTreeRoot and OpenAction
      1.3 Writing the IDtree
      1.4 Writing structure elements
      1.5 ParentTree
      1.6 Rolemap dictionary
      1.7 Classmap dictionary
      1.8 Namespaces
      1.9 Finishing the structure
      1.10 StructParents entry for Page
IV The \texttt{tagpdf-mc-shared} module
   Code related to Marked Content (mc-chunks), code shared by all modes
   Part of the tagpdf package
   1 Public Commands
   2 Public keys
   3 Marked content code – shared
      3.1 Variables and counters
      3.2 Functions
      3.3 Keys

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

1 Marked content code – generic mode
1.1 Variables ........................................ 75
1.2 Functions ........................................ 76
1.3 Looking at MC marks in boxes .................... 79
1.4 Keys ............................................. 87

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

1 Marked content code – luamode code ............... 89
1.1 Commands ......................................... 91
1.2 Key definitions ................................... 95

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

1 Public Commands ..................................... 98
2 Public keys .......................................... 99
  2.1 Keys for the structure commands ............... 99
  2.2 Setup keys ..................................... 101
3 Variables .......................................... 101
  3.1 Variables used by the keys ..................... 103
  3.2 Variables used by tagging code of basic elements 104
4 Commands .......................................... 104
  4.1 Initialization of the StructTreeRoot ............ 105
  4.2 Adding the /ID key ................................ 106
  4.3 Filling in the tag info .......................... 107
  4.4 Handlings kids .................................. 108
  4.5 Output of the object ........................... 111
5 Keys ............................................. 115
6 User commands ..................................... 121
7 Attributes and attribute classes .................... 129
  7.1 Variables ...................................... 129
  7.2 Commands and keys ............................. 129
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_start: will not restart tagging.

\tag_stop:n\{|label\}
\tag_start:n\{|label\}

The commands with argument allow to give a label. This is only used in debugging messages to allow to follow the nesting.

activate/spaces,activate/tree,activate/struct,activate/mc,activate/struct-dest

activate/struct-dest,activate/tree,activate/struct

Keys to activate the various tagging steps.

activate/struct-dest,activate/tree,activate/struct

activate/struct-dest,activate/tree,activate/struct

The key allows to suppress the creation of structure destinations

debug/log,activate/tagunmarked

data

debug/log

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

activate/tagunmarked,activate/tagunmarked

data

debug/log

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

page/tabsorder,tabsorder

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.

These are attributes used by the label/ref system.
1 Initialization and test if pdfmanagement is active.

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

<debug>
\@ifpackageloaded{tagpdf}{}\{\PackageWarning{tagpdf-debug}{tagpdf-not-loaded, quitting}\endinput\}
</debug>

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

\prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }

Debug mode has its special mapping:

\prop_gput:Nnn \g_msg_module_type_prop { tag / debug }{} 
\prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf-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

\ProvidesExplPackage {tagpdf-base} {2024-02-29} {0.98x}
{ part of tagpdf - provide base, no-op versions of the user commands }
</base>
3 Package options

There are only two documented options to switch for luatex between generic and luamode, TODO try to get rid of them. The option disabledelayedshipout is only temporary to be able to debug problem with the new shipout keyword if needed.

\bool_new:N \g__tag_mode_lua_bool
\bool_new:N \g__tag_delayed_shipout_bool
\bool_lazy_and:nNT
{ \bool_if_exist_p:N \l__pdfmanagement_delayed_shipout_bool }
{ \l__pdfmanagement_delayed_shipout_bool }
{ \bool_gset_true:N \g__tag_delayed_shipout_bool }
\DeclareOption {luamode} { \sys_if_engine_luatex:T { \bool_gset_true:N \g__tag_mode_lua_bool } }
\DeclareOption {genericmode} { \bool_gset_false:N \g__tag_mode_lua_bool }
\DeclareOption {disabledelayedshipout} { \bool_gset_false:N \g__tag_delayed_shipout_bool }
\ExecuteOptions{luamode}
\ProcessOptions

4 Packages

To be on the safe side for now, load also the base definitions
\RequirePackage{tagpdf-base}

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:

\AddToHook{begindocument}
{ \str_case:VnF \c_sys_backend_str
  { luatex } { \cs_new_protected:Npn \__tag_whatsits: {} }
  { dvisvgm } { \cs_new_protected:Npn \__tag_whatsits: {} }
}
\cs_new_protected:Npn \__tag_whatsits: { \tex_special:D {} }

4.1 a LastPage label

See also issue #2 in Accessible-xref
\__tag_lastpagelabel:
\exp_args:NNme \exp_args:NNe\iow_now:Nn \@auxout
{
  \token_to_str:N \new@label@record
  {\@tag@LastPage}
  {
    {abspage} \int_use:N \g_shipout_readonly_int
    {tagmcabs}\int_use:N \c@g__tag_MCID_abs_int
    {tagstruct}\int_use:N \c@g__tag_struct_abs_int
  }
}
\AddToHook{enddocument/afterlastpage}
{\_\_tag_lastpagelabel:}

(End of definition for \_\_tag_lastpagelabel:)

\begin{tabular}{l}
\l__tag_tmpa_tl\l__tag_tmpb_tl
\l__tag_get_tmpc_tl
\l__tag_get_parent_tmpa_tl
\l__tag_get_parent_tmpb_tl
\l__tag_tmpa_str
\l__tag_tmpa_prop
\l__tag_tmpa_seq
\l__tag_tmpb_seq
\l__tag_tmpa_clist
\l__tag_tmpa_int
\l__tag_tmpa_box
\l__tag_tmpb_box
\c__tag_property_mc_clist
\c__tag_property_struct_clist
\l__tag_loglevel_int
\end{tabular}

5 Variables

A few temporary variables
\tl_new:N \l__tag_tmpa_tl
\tl_new:N \l__tag_tmpb_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
\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 \_\_tag_tmpa_tl and others.)

Attribute lists for the label command. We have a list for mc-related labels, and one
for structures.
\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.)

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 \_\_tag_loglevel_int.)
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 \texttt{\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.

\begin{verbatim}
\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{verbatim}

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.

\begin{verbatim}
\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{verbatim}

\begin{verbatim}
\bool_new:N \g__tag_tagunmarked_bool
\end{verbatim}

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.

\begin{verbatim}
\bool_new:N \g__tag_tagunmarked_bool
\end{verbatim}

\section{Variants of \texttt{l3} commands}

\begin{verbatim}
\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{Nne} %** unneeded
\cs_generate_variant:Nn \str_set_convert:Nnnn {Nonn, Noon, Nnon }
\cs_generate_variant:Nn \clist_map_inline:nn {on}
\end{verbatim}
7 Label and Reference commands

To ease transition to properties we setup internal definition. They can be replaced by the property definitions once that is released. ** do it!

At first a command to define new properties
\begin{verbatim}
\cs_new_eq:NN \__tag_property_new:nnnn \property_new:nnnn
\end{verbatim}

For the non-shipout code we need also the option to reset property
\begin{verbatim}
\cs_new_eq:NN \__tag_property_gset:nnnn \property_gset:nnnn
\end{verbatim}

The command to reference while giving a local default.
\begin{verbatim}
\cs_new_eq:NN \__tag_property_ref:nnn \property_ref:nnn
\cs_new_eq:NN \__tag_property_ref:nn \property_ref:nn
\end{verbatim}

The command to record
\begin{verbatim}
\cs_new_protected:Npn \__tag_property_record:nn #1#2
{ \@bsphack \property_record:nn{#1}{#2} \@esphack }
\end{verbatim}

And a few variants
\begin{verbatim}
\cs_generate_variant:Nn \__tag_property_ref:nnn {enn}
\cs_generate_variant:Nn \__tag_property_ref:nn {en}
\cs_generate_variant:Nn \__tag_property_record:nn {en,eV}
\end{verbatim}

\begin{verbatim}
\__tag_property_ref_lastpage:nn
\cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
{ \__tag_property_ref:nnn {@tag@LastPage}{#1}{#2} }
\end{verbatim}

\begin{verbatim}
\__tag_property_new:nnnn
\__tag_property_gset:nnnn
\__tag_property_ref:nnn
\__tag_property_ref_lastpage:nn
\end{verbatim}

A command to retrieve the lastpage label, this will be adapted when there is a proper, kernel lastpage label.
\begin{verbatim}
\cs_new:Npn \__tag_property_ref_lastpage:nn #1 #2
{ \__tag_property_ref:nnn {@tag@LastPage}{#1}{#2} }
\end{verbatim}

8 Setup label attributes

This are attributes used by the label/ref system. With structures we store the structure number \texttt{tagstruct} and the object reference \texttt{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 \texttt{tagabspage}, the absolute id \texttt{tagmcabs}, and the id on the page \texttt{tagmcid}.

\begin{verbatim}
\__tag_property_new:nnnn
\{ tagstruct \} \{ now \}
\{0\} \{ \int_use:N \c@g__tag_struct_abs_int \}
\__tag_property_new:nnnn \{ tagstructobj \} \{ now \} \{ \}
\end{verbatim}
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.

10 General tagging commands

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.
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.

\begin{verbatim}
\l__tag_tag_stop_int
\cs_set_protected:Npn \tag_stop:
\{
  \msg_note:nnx {tag / debug }{tag-stop}{ \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:
  }
  \msg_note:nnx {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }
\}
\cs_set_protected:Npn \tag_stop:n #1
\{
  \msg_note:nnxx {tag / debug }{tag-stop}{ \int_use:N \l__tag_tag_stop_int }{#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_protected:Npn \tag_start: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:
  }
  \msg_note:nnxx {tag / debug }{tag-start}{ \int_use:N \l__tag_tag_stop_int }{#1}
\}
\end{verbatim}

\end{document}
11 Keys for tagpdfsetup

TODO: the log-levels must be sorted

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

Subkeys/values are defined in various other places.

The \texttt{log} takes currently the values \texttt{none}, \texttt{v}, \texttt{vv}, \texttt{vvv}, \texttt{all}. The description of the log levels is in tagpdf-checks.
263 debug/log / vvv .code:n = {\int_set:Nn \l__tag_loglevel_int { 3 }},
264 debug/log / all .code:n = {\int_set:Nn \l__tag_loglevel_int { 10 }},
265 debug/uncompress .code:n = { \pdf_uncompress: },

deprecated but still needed as the documentmetadata key argument uses it.
266 log .meta:n = {debug/log={#1}},
267 uncompress .code:n = { \pdf_uncompress: },

(End of definition for debug/log (setup-key) and others. These functions are documented on page 6.)

activate/tagunmarked (setup-key)
tagunmarked (deprecated)
This key allows to set if (in luamode) unmarked text should be marked up as artifact. The initial value is true.
268 activate/tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,
269 activate/tagunmarked .initial:n = true,

deprecated name
tagunmarked .bool_gset:N = \g__tag_tagunmarked_bool,

(End of definition for activate/tagunmarked (setup-key) and tagunmarked (deprecated). These functions are documented on page 6.)

page/tabsorder (setup-key)
tabsorder (deprecated)
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.
271 page/tabsorder .choice:,
272 page/tabsorder / row .code:n = \pdfmanagement_add:nnn { Page } {Tabs}{/R},
273 page/tabsorder / column .code:n = \pdfmanagement_add:nnn { Page } {Tabs}{/C},
274 page/tabsorder / structure .code:n = \pdfmanagement_add:nnn { Page } {Tabs}{/S},
275 page/tabsorder / none .code:n = \pdfmanagement_remove:nn {Page} {Tabs},
276 page/tabsorder .initial:n = structure,

deprecated name
tabsorder .meta:n = {page/tabsorder={#1}},
279 }

(End of definition for page/tabsorder (setup-key) and tabsorder (deprecated). These functions are documented on page 6.)

12 loading of engine/more dependent code

\sys_if_engine_luatex:T
283 {\file_input:n {tagpdf-luatex.def}}
285 {/package}
286 {mcloading}
287 \bool_if:NTF \g__tag_mode_lua_bool
288 {\RequirePackage {tagpdf-mc-code-lua}}
\RequirePackage {tagpdf-mc-code-generic} \\
\bool_if:NTF \g__tag_mode_lua_bool \\
{ \\
\RequirePackage {tagpdf-debug-lua} \\
} \\
{ \\
\RequirePackage {tagpdf-debug-generic} \\
} \\
(/debug)
Part I

The tagpdf-checks module

Messages and check code

Part of the tagpdf package

1 Commands

\tag_if_active_p: * \tag_if_active:TF *

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

\tag_get:n{⟨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:N{⟨box⟩} \tag_if_box_tagged:N{⟨box⟩} \tag_if_box_tagged:NTF *

This tests if a box contains tagging commands. It relies currently on that the code, that saved the box, correctly sets the command \_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

<table>
<thead>
<tr>
<th>Argument</th>
<th>type</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ShowTaggingmc-data = num</td>
<td>log+term</td>
<td>lua-only</td>
</tr>
<tr>
<td>\ShowTaggingmc-current</td>
<td>log+term</td>
<td></td>
</tr>
<tr>
<td>\ShowTaggingstruck-stack= [log</td>
<td>show]</td>
<td>log or term+stop</td>
</tr>
<tr>
<td>\ShowTaggingdebug/structures = num</td>
<td>log+termn</td>
<td>debug mode only</td>
</tr>
</tbody>
</table>

17
2.2 Messages in checks and commands

<table>
<thead>
<tr>
<th>command</th>
<th>message</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@_check_structure_has_tag:n</td>
<td>struct-missing-tag</td>
<td>error</td>
</tr>
<tr>
<td>@@_check_structure_tag:N</td>
<td>role-unknown-tag</td>
<td>warning</td>
</tr>
<tr>
<td>@@_check_info_closing_struct:n</td>
<td>struct-show-closing</td>
<td>info</td>
</tr>
<tr>
<td>@@_check_no_open_struct:</td>
<td>struct-faulty-nesting</td>
<td>error</td>
</tr>
<tr>
<td>@@_check_add_tag_role:nn</td>
<td>role-unknown</td>
<td>warning, info (&gt;0), warning</td>
</tr>
<tr>
<td>@@_check_no_open_struct:</td>
<td>struct-faulty-nesting</td>
<td>error</td>
</tr>
<tr>
<td>@@_check_mc_if_nested:</td>
<td>mc-nested</td>
<td>info (2), info+seq_log (&gt;2)</td>
</tr>
<tr>
<td>@@_check_mc_pushed_popped:nn</td>
<td>mc-tag-missing, role-unknown-tag</td>
<td>error (missing), warning (unknown)</td>
</tr>
<tr>
<td>@@_check_mc_used:n</td>
<td>mc-used-twice</td>
<td>warning</td>
</tr>
<tr>
<td>@@_check_show_MCID_by_page:</td>
<td>currently unused</td>
<td></td>
</tr>
<tr>
<td>\tag_mc_use:n</td>
<td>mc-label-unknown, mc-used-twice</td>
<td>warning</td>
</tr>
<tr>
<td>\role_add_tag:nn</td>
<td>new-tag</td>
<td>info (&gt;0)</td>
</tr>
<tr>
<td>@@_struct_write_obj:n</td>
<td>struct-no-objnum</td>
<td>error</td>
</tr>
<tr>
<td>@@_struct_write_obj:n</td>
<td>struct-orphanspace</td>
<td>warning</td>
</tr>
<tr>
<td>\tag_struct_insert_annot:nn</td>
<td>struct-faulty-nesting</td>
<td>error</td>
</tr>
<tr>
<td>\tag_struct_begin:n</td>
<td>struct-faulty-nesting</td>
<td>error</td>
</tr>
<tr>
<td>\tag_struct_used:n</td>
<td>struct-unknown</td>
<td>warning</td>
</tr>
<tr>
<td>@@_tree_fill_parenttree:</td>
<td>tree-mcid-index-wrong</td>
<td>warning</td>
</tr>
<tr>
<td>in enddocument/info-hook:</td>
<td>para-hook-count-wrong</td>
<td>warning</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>message</th>
<th>log-level</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>WARN TAG-NOT-TAGGED:</td>
<td>1</td>
<td>shouldn’t happen</td>
</tr>
<tr>
<td>WARN TAG-OPEN-MC:</td>
<td>1</td>
<td>shouldn’t happen</td>
</tr>
<tr>
<td>WARN SHIPOUT-MC-OPEN:</td>
<td>1</td>
<td>e.g. from empty hbox</td>
</tr>
<tr>
<td>WARN SHIPOUT-UPS:</td>
<td>0</td>
<td>shouldn’t happen</td>
</tr>
<tr>
<td>WARN TEX-MC-INSERT-MISSING:</td>
<td>0</td>
<td>shouldn’t happen</td>
</tr>
<tr>
<td>WARN TEX-MC-INSERT-NO-KIDS:</td>
<td>2</td>
<td>e.g. from empty hbox</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>message</th>
<th>log-level</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO SHIPOUT-INSERT-LAST-EMC</td>
<td>3</td>
<td>finish of shipout code</td>
</tr>
<tr>
<td>INFO SPACE-FUNCTION-FONT</td>
<td>3</td>
<td>interwordspace code</td>
</tr>
<tr>
<td>INFO TAG-ABSPAGE</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-ARGS</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-ENDHEAD</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-HEAD</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-INSERT-ARTIFACT</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
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 | replaced by artifact
INFO TAG-NOT-TAGGED | 2 | 
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.

<table>
<thead>
<tr>
<th>command name</th>
<th>action</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>\tag_mc_begin:n</td>
<td>mc-begin-insert</td>
<td>msg</td>
</tr>
<tr>
<td></td>
<td>mc-begin-ignore</td>
<td>msg</td>
</tr>
</tbody>
</table>

### 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

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

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 is unknown.

role-missing
role-unknown
role-unknown-tag
role-unknown-NS
role-tag
new-tag
role-parent-child
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.

\ProvidesExplPackage {tagpdf-checks-code} {2024-02-29} {0.98x}
{part of tagpdf - code related to checks, conditionals, debugging and messages}
3 Messages

3.1 Messages related to mc-chunks

mc-nested This message is issued if 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.

\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} { required-tag-missing---mcid=#1 }

(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 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.\\n  It-is-turned-into-an-artifact.\\n  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\textit{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.)
Message issued at the end if there are beside Root other open structures on the stack.

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

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

The following messages are only needed in debug mode.

This two messages are used to show the current structures in the log and terminal.

```latex
\msg_new:nnn \{ tag/debug \} \{ show-struct \}
\{
  \text{The-structure-#1-}
  \text{\tl_if_empty:nTF \{#2\}}
  \text{\{ is-empty \} is-#2 \text{contains: \#2 \}}
  \text{\}}
\}
\msg_new:nnn \{ tag/debug \} \{ show-kids \}
\{
  \text{The-structure-has-the-following-kids:}
  \text{\tl_if_empty:nTF \{#2\}}
  \text{\{ \} is-#2 \}}
  \text{\}}\text{ }
```

(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.

```latex
\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

Warning message if either the tag or the role is missing

```latex
\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} \}
```
role-parent-child

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

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

(role-parent-child. This function is documented on page 20.)

role-remapping

This is info and warning message about role-remapping

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

(role-remapping. This function is documented on page 20.)

role-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 }
```

(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}
```

(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}
```

(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⟩}
```

(para-hook-count-wrong. This function is documented on page 20.)
4 Retrieving data

\texttt{\textbackslash tag\_get:n} This retrieves some data. This is a generic command to retrieve data. Currently the only sensible values for the argument are \texttt{mc\_tag}, \texttt{struct\_tag} and \texttt{struct\_num}.

\begin{verbatim}
\{\texttt{\use:c{\_tag\_get\_data\_#1:}\}}
\end{verbatim}

(End of definition for \texttt{\textbackslash tag\_get:n}. This function is documented on page 17.)

5 User conditionals

\texttt{\textbackslash tag\_if\_active:p, \textbackslash 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.

\begin{verbatim}
\prg_set_conditional:Npnn \tag_if_active: { p , T , TF, F }
{ \prg_return_true: }
\end{verbatim}

(End of definition for \texttt{\textbackslash tag\_if\_active:TF}. This function is documented on page 17.)

\texttt{\textbackslash tag\_if\_box\_tagged:p, \textbackslash 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 \texttt{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.

\begin{verbatim}
\{\texttt{\_tag\_get\_data\_#1:}\}
\end{verbatim}

(End of definition for \texttt{\textbackslash tag\_get\_data}. 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

This checks if mc are active.

\__tag_check_if_active_mc:TF
\__tag_check_if_active_struct:TF

\cs_new_protected:Npn \__tag_check_if_active_mc:TF #1 {TF}
\bool_lazy_and:nnTF { \g__tag_active_mc_bool } { \l__tag_active_mc_bool }
{ \prg_return_true: }
{ \prg_return_false: }

\cs_new_protected:Npn \__tag_check_if_active_struct:TF #1 {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

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

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

(End of definition for \__tag_check_structure_has_tag:n \#1 %#1 struct num)
\__tag_check_structure_tag:N
This checks if the name of the tag is known, either because it is a standard type or has been rolemapped.
\cs_new_protected:Npn \__tag_check_structure_tag:N #1
\{
\prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
\{
\msg_warning:nne { tag } {role-unknown-tag} {#1}
\}
\}
\end{definition}
\__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{definition}
\__tag_check_no_open_struct:
This checks if there is an open structure. It should be used when trying to close a structure. It errors if false.
\cs_new_protected:Npn \__tag_check_no_open_struct:
\{
\msg_error:nn { tag } {struct-faulty-nesting}
\}
\end{definition}
\__tag_check_struct_used:n
This checks if a stashed structure has already been used.
\cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
\{
\prop_get:cnNT \g__tag_struct_\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop
\{P\}
\l__tag_tmpa_tl
\{
\msg_warning:nnn { tag } {struct-used-twice} {#1}
\}
\end{definition}
6.3 Checks related to roles

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
\begin{verbatim}
\tl_if_empty:nTF {#2}
\begin{verbatim}
\msg_error:nnn { tag } {role-missing} {#1}
\end{verbatim}
\begin{verbatim}
\prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l_tmpa_tl
\begin{verbatim}
\int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
\begin{verbatim}
\msg_info:nnnn { tag } {role-tag} {#1} {#2}
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\msg_error:nnn { tag } {role-unknown} {#2}
\end{verbatim}
\end{verbatim}
\end{verbatim}
\end{verbatim}

Similar with a namespace

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

(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:
\begin{verbatim}
\tl_if_empty:nTF {\__tag_mc_if_in:T}
\begin{verbatim}
\msg_error:nnn { tag } {role-missing} {#1}
\end{verbatim}
\begin{verbatim}
\prop_get:NnNTF \g__tag_role_tags_NS_prop {#2} \l_tmpa_tl
\begin{verbatim}
\int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
\begin{verbatim}
\msg_info:nnnn { tag } {role-tag} {#1} {#2}
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\msg_error:nnn { tag } {role-unknown} {#2}
\end{verbatim}
\end{verbatim}
\end{verbatim}

(End of definition for \__tag_check_add_tag_role:nn.)
\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: }
 }
}
\__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 }
}
\__tag_check_mc_tag:N
This checks if the mc has a (known) tag.
\cs_new_protected:Npn \__tag_check_mc_tag:N #1 %#1 is var with a tag name in it
{ \tl_if_empty:NT #1
 { \msg_error:nne { tag } {mc-tag-missing} { \__tag_get_mc_abs_cnt: }
 }
 \prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
 { \msg_warning:nne { tag } {role-unknown-tag} {#1}
 }
}
\__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:
\}
\__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}
\int_compare:nNnT
\{
\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}
\}
\}
\}
\}
\__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_tmpa_seq
\int_step_inline:nnnn
\{
\}
\}
\{
\__tag_property_ref_lastpage:nn
{tagmcabs}
{-1}
\}
\{

(End of definition for \g__tag_check_mc_used_intarray and \__tag_check_init_mc_used:.)

(End of definition for \__tag_check_mc_used:n.)

(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.

\int_compare:nT
\{\_tag_property_ref:enn
{mcid-####1}
{tagabspage}
{-1}
=
###1
\}
\seq_gput_right:Ne \l_tmpa_seq
{Page##1-####1-
\_tag_property_ref:enn
{mcid-####1}
{tagmcid}
{-1}
}
\seq_show:N \l_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.

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_if_mc_in_galley:TF)

This checks if a extra top mark (“extra-tmb”) is needed. According to the analysis this 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_sc_tmb_missing: { T,F,TF }
{\bool_if:nTF
\begin{verbatim}
{| |
\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:Nnnpnn \_\_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:nnnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:~\tl_to_str:n{#2}~[\msg_line_context:} }
\msg_new:nnnn { 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:nnnn { tag / debug } {mc-end} {inserted} }
}
\cs_new_protected:Npn \_\_tag_debug_mc_end_ignore:
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
{ \msg_note:nnnn { tag / debug } {mc-end} {inserted} }
}
\cs_new_protected:Npn \_\_tag_debug_mc_end_insert:
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
{ \msg_note:nnnn { tag / debug } {mc-end} {inserted} }
}
\cs_new_protected:Npn \_\_tag_debug_mc_end_ignore:
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
{ \msg_note:nnnn { tag / debug } {mc-end} {inserted} }
}
\end{verbatim}
And now something for the structures
\msg_new:n { tag / debug } {struct-begin}
{ Struct-\tag_get:n{struct_num}-begin-#1-with-options:-\tl_to_str:n(#2)-\msg_line_context: }
\msg_new:n { tag / debug } {struct-end}
{ Struct-end-#1-\msg_line_context: }
\msg_new:n { 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:nnn { tag / debug } {struct-begin} {inserted} { #1 }
\seq_log:N \g__tag_struct_tag_stack_seq
}
\cs_new_protected:Npn \__tag_debug_struct_beginIgnore:n #1
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
\msg_note:nnn { 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_endIgnore:
{ \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_tempa_tl
}
\str_if_eq:eeF
\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl
{
\msg_warning:nnee { tag/debug }{ struct-end-wrong }
\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl
}

This tracks tag stop and start. The tag-stop message should go before the int is increased.
The tag-start message after the int is decreased.
\msg_new:nnn { tag / debug } {tag-stop}
{
\int_if_zero:nTF
\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl
{
{Tagging-stopped}
{Tagging-(not)-stopped-(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-start}
{
\int_if_zero:nTF
\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl
{
{Tagging-restarted}
{Tagging-(not)-restarted}\\level: \int_eval:n(#1+1)-\int_eval:n(#1)\tl_if_empty:nF{-#2}{,-label:-#2}-\msg_line_context:\}

(/debug)
Part II

The tagpdf-user module
Code related to \LaTeX{} 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 \tag_tool:n{⟨key val⟩}

\tagtool

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

These are wrappers around \tagmcbegin:n, \tagmcend:n and \tagmcuse:n. The commands and their argument are documented 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 \tagmc_if_in:TF. and tests if an mc is open or not. It is mostly of importance for pdflatex as lualatex doesn’t mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

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

\tagstructbegin \tagstructbegin \{\langle\text{key-val}\rangle\}\tagstructend
\tagstructend \tagstructuse\{\langle\text{label}\rangle\}\tagstructuse

These are direct wrappers around \tagstructbegin:n, \tagstructend: and \tagstructuse:n. The commands and their argument are documented in the tagpdf-struct module.

4 Debugging

\ShowTagging \ShowTagging \{\langle\text{key-val}\rangle\}\ShowTagging

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\{\langle\text{show-key}\rangle\} \mc-data = \langle\text{number}\rangle

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\{\langle\text{show-key}\rangle\} \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\{\langle\text{show-key}\rangle\} \mc-marks = \langle\text{show|use}\rangle

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

\struct-stack\{\langle\text{show-key}\rangle\} \struct-stack = \langle\text{log|show}\rangle

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

\debug/structures\{\langle\text{show-key}\rangle\} \debug/structures = \langle\text{structure number}\rangle

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

\pdffakebox
(lua-only) This provides a lua-version of the \pdffakebox 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.

\tagpdfsupressmarks
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\tagpdfsupressmarks{\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 \( \text{(setup-key)} \) 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:

\pdannot dict put:nnn
\{ link/GoTo \}
\{ Contents \}
\{ (ref) \}

6 Socket support

\\tag_socket_use:n \tag_socket_use:n \{(socket name)\}
\tag_socket_use:nn \tag_socket_use:nn \{(socket name)\} \{(socket argument)\}
\UseTaggingSocket \UseTaggingSocket \{(socket name)\} \{(socket argument)\}

The next \LaTeX\ (2024-06-01) will use special sockets for the tagging.
These sockets will use names starting with tagsupport/. Usually, these sockets have exactly two plugs defined: noop (when no tagging is requested or tagging is not wanted for some reason) and a second plug that enables the tagging. There may be more, e.g.,
tagging with special debugging, etc., but right now it is usually just on or off.

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 the socket name is specified without tagsupport/, i.e.,

\UseTaggingSocket{foo} \rightarrow \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.
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`.

The L3 programming layer versions `\tag_socket_use:n` and `\tag_socket_use:nn` 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

8 Setup and preamble commands

9 Commands for the mc-chunks
\section*{10 Commands for the structure}

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

\begin{verbatim}
\NewDocumentCommand \tagstructbegin { m } {
  \tag_struct_begin:n {#1}
}
\NewDocumentCommand \tagstructend { } {
  \tag_struct_end:
}
\NewDocumentCommand \tagstructure { m } {
  \tag_struct_use:n {#1}
}
\end{verbatim}

\textit{(End of definition for \tagstructbegin, \tagstructend, and \tagstructure. These functions are documented on page 36.)}
11 Socket support

Until we can be sure that the kernel defines the commands we provide them before redefining them:

\providecommand\tag_socket_use:n[1]{}
\providecommand\tag_socket_use:nn[2]{}
\providecommand\UseTaggingSocket[1]{}

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

\ERRORusetaggingsocket

(End of definition for \tag_socket_use:n, \tag_socket_use:nn, and \UseTaggingSocket. These functions are documented on page 38.)

12 Debugging

This is a generic command for various show commands. It takes a keyval list, the various keys are implemented below.
`\keys_set:nn { __tag / show }{ #1}
}

(End of definition for `ShowTagging`. This function is documented on page 36.)

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

(End of definition for `mc-data (show-key)`. This function is documented on page 36.)

**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
    {
      \sys_if_engine_luatex:T
      {
        \int_compare:nNnTF
        { -2147483647 }
        =
        {
          \lua_now:e
          {
            \text.print
            (\text.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
              }
              \textio.write_nl("=")
            }
          }
        }
      }
      {
        \lua_now:e
        {
          \ltx.__tag.trace.log
          {
            "mc-current:~-no-MC-open,-current-abscnt
            =\__tag_get_mc_abs_cnt:"
            ,0
          }
          \textio.write_nl("=")
        }
      }
    }
  }
}
```

42
mc-current (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
  }
}
\__tag_check_if_mc_tme_missing:T
{
\io_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 36.)

\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
}

(struct-stack (show-key). This function is documented on page 36.)

\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 }
}

(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.

(End of definition of debug/structures (show-key). This function is documented on page 36.)
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.

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
}
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_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

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

```
\providecommand\pdffakespace{}
```

(End of definition for \pdffakespace. This function is documented on page 37.)

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_main_tag_tl \l__tag_para_attr_class_tl \l__tag_para_main_attr_class_tl
```

At first some variables.

```
//package
\bool_new:N \l__tag_para_flattened_bool
\bool_new:N \l__tag_para_bool
\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
\int_new:N \g__tag_para_main_struct_tl
\int_new:N \g__tag_para_tag_default_tl
\int_new:N \g__tag_para_main_tag_tl
\int_new:N \g__tag_para_attr_class_tl
\int_new:N \g__tag_para_main_attr_class_tl
```

(End of definition for \g__tag_root_default_tl, activate (setup-key), and activate/socket (setup-key). These functions are documented on page 35.)
this will hold the structure number of the current text-unit.

```latex
\l_new:N \g__tag_para_main_struct_tl
\l_new:N \l__tag_para_tag_default_tl
\l_set:Nn \l__tag_para_tag_default_tl { text }
\l_new:N \l__tag_para_tag_tl
\l_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }
\l_new:N \l__tag_para_main_tag_tl
\l_set:Nn \l__tag_para_main_tag_tl { text-unit }
```

this is perhaps already defined by the block code

```latex
\l_if_exist:NF \l__tag_para_attr_class_tl
{ \l_new:N \l__tag_para_attr_class_tl }
\l_new:N \l__tag_para_main_attr_class_tl
```

(End of definition for \l__tag_para_bool and others.)

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

```latex
\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.)

```latex
\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 }
```

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

```latex
\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: 
{ }
```
We want to be able to inspect the current para main structure, so we need a command to store its structure number.

TEMPORARILY FIX (2023-11-17). Until latex-lab is updated we must adapt a sec command:

```latex
\AddToHook{package/latex-lab-testphase-sec/after}
{\cs_set_protected:Npn \@kernel@tag@hangfrom #1
{\tagstructbegin{tag=\l__tag_para_tag_tl}
\__tag_gincr_para_begin_int:
\tagstructbegin{tag=Lbl}
\setbox\@tempboxa
\hbox
{\bool_lazy_and:nnT{\tag_if_active_p:}{\g__tag_mode_lua_bool}{\tagmcbegin{tag=Lbl}{#1}}}
\tag_stop:n{hangfrom}
\hangindent \wd\@tempboxa\noindent
\tagmcbegin{}\box\@tempboxa\tagmce\tagstructend\tagmcbegin{}
}

\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}
\tl_set:Nn \l__tag_struct_abs_tl \int_use:N \c@g__tag_struct_abs_int}
```

and one temporary adaptions 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}
}```
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 }
  \keys_define:nn { tag / tool }
  \keys_define:nn { __tag / setup }
  \keys_define:nn { tag / tool }
```

(End of definition for para/tagging (setup-key) and others. These functions are documented on page 37.)

Helper command for debugging:

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

49
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.
there was no real difference between the original and in the block variant, only a debug message. We therefore define only a plain variant.

By default we assign the plain plug:

```
\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_struct_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:
    }
  }
}
```

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

We need at least the new-or-1 code. In generic mode we also must insert the code to finish the MC-chunks.
This two command switch para mode on and off. \tagpdfsetup could be used too but is longer. An alternative is \tagtool\{\paras\false\}. These functions are documented on page 37.

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.

\begin{verbatim}
\@hangfrom
\{
  \tagstructbegin{tag=H1}\%
  \tagmcbegin {tag=H1}\%
  #2
  \}
  \tagpdfsuppressmarks{\tagmcend}\tagstructend\%
  \NewDocumentCommand\tagpdfsuppressmarks{m}
  {{\use:c{\__tag_mc_disable_marks:} #1}}
\end{verbatim}

(End of definition for \tagpdfsuppressmarks. This function is documented on page 37.)
13.5 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.

\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:}
}
}
\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_stop:n{headfoot}
}
\cs_new_protected:Npn \__tag_exclude_headfoot_end:
{
\tag_start: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
\bool_gset_false:N \g__tag_in_mc_bool
}
\tag_stop:n{headfoot}
}
\__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_stop:n{headfoot}
\}
\cs_new_protected:Npn \__tag_exclude_struct_headfoot_end:  
\{ 
\tag_start: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
\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 {ps}}
\cs_set:Nn \__tag_hook_kernel_before_foot: { \__tag_exclude_struct_headfoot_begin:n {ps}}
\cs_set:Nn \__tag_hook_kernel_after_head: \__tag_exclude_struct_headfoot_end:
\cs_set: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 38.)

13.6 Links

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

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

\hook_gput_code:nnn
{pdfannot/link/URI/after}
{tagpdf}
{
  \tag_struct_insert_annotee {\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}
{
% "alternative descriptions " for PAX3. How to get better text here??
\pdfannot_dict_put:nnn
{ link/URI }
{ Contents }
{ (url) }
\pdfannot_dict_put:nnn
{ link/GoTo }
{ Contents }
{ (ref) }
</package>
Part III
The tagpdf-tree module
Commands trees and main
dictionaries
Part of the tagpdf package

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.

1.1 Check structure

\_\_tag_tree_final_checks:

(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.

This is the object for the root object, the StructTreeRoot

\pdf_object_new:n { __tag/struct/0 }  
(End of definition for __tag/struct/0.)

We need a variable that indicates which structure is wanted in the OpenAction. By default we use 1 (the Document structure).

\tl_new:N \g__tag_tree_openaction_struct_tl  \tl_gset:Nn \g__tag_tree_openaction_struct_tl {1}  
(End of definition for \g__tag_tree_openaction_struct_tl.)

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/startpage (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  
\seq_set_split:NnV \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 \c_space_tl /SD-{\pdf_object_ref:e{__tag/struct/\g__tag_tree_openaction_struct_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.
there should be always a \texttt{/Fit etc} in the array but better play safe here ... 
\begin{verbatim}
\int_compare:nNnTF{ \seq_count:N \l__tag_tmpa_seq } > {1} 
{ /\seq_item:Nn\l__tag_tmpa_seq{2} } \end{verbatim}

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

\begin{verbatim}
\g__tag_tree_id_pad_int \int_new:N \g__tag_tree_id_pad_int 
(End of definition for \texttt{\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} }
\end{verbatim}

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.

\begin{verbatim}
\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:nn {\c@g__tag_struct_abs_int} 
\{ \int_incr:N \l__tag_tmpa_int \int_step_inline:nn {\c@g__tag_struct_abs_int} 
\{ \int_incr:N \l__tag_tmpa_int \tl_put_right:Ne \l__tag_tmpa_tl \} \end{verbatim}
1.4 Writing structure elements

The following commands are needed to write out the structure.

\_\_tag\_write\_struct\_treeroot:

This writes out the root object.

\cs\_new\_protected:Npn \_\_tag\_write\_struct\_treeroot:

{ { \_\_tag\_prop\_gput:cne
  \{ g\_tag\_struct\_0\_prop \}
  \{ ParentTree \}
  \{ \pdf\_object\_ref:n \{ \_\_tag\_tree\_parenttree \} \}
  \_\_tag\_prop\_gput:cne
  \{ g\_tag\_struct\_0\_prop \}
  \{ RoleMap \}
  \{ \pdf\_object\_ref:n \{ \_\_tag\_tree\_rolemap \} \}
  \_\_tag\_struct\_fill\_kid\_key:n { 0 }
  \_\_tag\_struct\_get\_dict\_content:nN { 0 } \l\_\_tag\_tmpa\_tl
  \pdf\_object\_write:nne
  \{ \_\_tag\_struct\_0\}
  \{ dict \}
  \{ \l\_\_tag\_tmpa\_tl \}
\__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.

\newcounter { 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
\}
\}

62
\l__tag_parenttree_content_tl

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

\l_new:N \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 clear if labels are needed. See lua code
\int_step_inline:nnnn{1}{1}{\__tag_property_ref_lastpage:nn{tagmcabs}{-1}}
\int_compare:nT
\int_eval:n {##1-1}
\int_compare:nTF \l__tag_tmpa_prop {####1} \l__tag_tmpa_tl
\pdf_object_if_exist:eTF { __tag/struct/\l__tag_tmpa_tl }
\pdf_object_ref:e { __tag/struct/\l__tag_tmpa_tl }
null
\c_space_tl
\cs_set_protected:Npn \__tag_tree_parenttree_rerun_msg: {63}
\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:NV \l__tag_parenttree_content_tl\g__tag_parenttree_objr_tl
\pdf_object_write:nne {\__tag/tree/parenttree }{dict}
{\ltx.__tag.func.output_parenttree
(\int_use:N\g_shipout_readonly_int
))
}
{\l__tag_parenttree_content_tl\c_space_tl}[1]
}
)
(End of definition for \__tag_tree_write_parenttree:)

64
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}}

\{}

\pdf\dict\_gput:nne \{g\_tag\_role\_RoleMap\_dict\}

\{\{##1\}

\}

\prop\_name\_from\_unicode\_e:n{##2}\}

\}

\pdf\object\_write:nne \{ \__tag\_tree\_rolemap \}{dict}

\{}

\pdf\dict\_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\_tmap\_tl

\seq\_remove\_duplicates:N \g\_tag\_attr\_class\_used\_seq

\seq\_set\_map:NNn \l\_tag\_tmap\_seq \g\_tag\_attr\_class\_used\_seq

\{}

\#1\c\_space\_tl

\{}

\prop\_item:Nn \g\_tag\_attr\_entries\_prop
1.8 Namespaces

Namespaces are handled 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.)
1.9 Finishing the structure

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

\__tag_finish_structure:

\begin{verbatim}
\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:
\__tag_tree_write_parenttree:
\__tag_tree_write_idtree:
\__tag_tree_write_rolemap:
\__tag_tree_write_classmap:
\__tag_tree_write_namespaces:
\__tag_tree_write_structelements: %this is rather slow!!
\__tag_tree_write_structtreeroot:
}
}
\end{verbatim}

(End of definition for \__tag_tree_write_namespaces.)

1.10 StructParents entry for Page

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

\begin{verbatim}
\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 } }
}
}
\end{verbatim}

(End of definition for \__tag_finish_structure.)
Part IV

The tagpdf-mc-shared module
Code related to Marked Content (mc-chunks), code shared by all modes

Part of the tagpdf package

1 Public Commands

\tag mc begin:n \tag mc begin:n{⟨key-values⟩}
\tag mc end:
\tag mc end:

These commands insert the end code of the marked content. They don’t end a group and
in generic mode it doesn’t matter if they are in another group as the starting commands.
In generic mode both commands check if they are correctly nested and issue a warning
if not.

\tag mc use:n \tag mc use:n{⟨label⟩}

These command allow to record a marked content that was stashed away before into the
current structure. A marked content can be used only once – the command will issue a
warning if an mc is use a second time.

\tag mc artifact group begin:n \tag mc artifact group begin:n {⟨name⟩}
\tag mc artifact group end: \tag mc artifact group end:

New: 2019-11-20
This command pair creates a group with an artifact marker at the begin and the end.
Inside the group the tagging commands are disabled. It allows to mark a complete
region as artifact without having to worry about user commands with tagging commands.
⟨⟨name⟩⟩ should be a value allowed also for the artifact key. It pushes and pops mc-chunks
at the begin and end. TODO: document is in tagpdf.tex

\tag mc end push: \tag mc end push:
\tag mc begin pop:n \tag mc begin pop:n{⟨key-values⟩}

Rev: 2021-04-22
If there is an open mc chunk, \tag mc end push: ends it and pushes its tag of the
(global) stack. If there is no open chunk, it puts −1 on the stack (for debugging) \tag mc begin pop:n removes a value from the stack. If it is different from −1 it opens a tag
with it. The reopened mc chunk looses info like the alt text for now.

\tag mc if in:TF \tag mc if in:TF{⟨true code⟩} {⟨false code⟩}

Determines if a mc-chunk is open.

68
\texttt{\textbackslash tag\_mc\_reset\_box:N} \texttt{\{}(box)\texttt{\}}

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

2 Public keys

The following keys can be used with \texttt{\textbackslash tag\_mc\_begin:n}, \texttt{\textbackslash tagmcbegin}, \texttt{\textbackslash tag\_mc\_begin\_pop:n},

\texttt{\textbackslash tag\_key}

This key is required, unless artifact is used. The value is a tag like \texttt{P} or \texttt{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 \texttt{H4} is fine).

\texttt{\textbackslash artifact\_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 \texttt{pagination, layout, page, background} and \texttt{notype} (this is the default).

\texttt{\textbackslash raw\_key}

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

\texttt{\textbackslash alt\_key}

This key inserts an \texttt{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.

\texttt{\textbackslash actualtext\_key}

This key inserts an \texttt{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.

\texttt{\textbackslash label\_key}

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

\texttt{\textbackslash stash\_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 splitted into three parts: code shared by all engines, code specific to luamode and code not used by luamode.
3 Marked content code – shared

3.1 Variables and counters

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

\g__tag_MCID_abs_int

\__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 }

\__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.
\cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }

\g__tag_in_mc_bool This booleans record if a mc is open, to test nesting.
\bool_new:N \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

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:
\begin{verbatim}
\seq_new:N \g__tag_mc_stack_seq
\end{verbatim}
\textit{(End of definition for \texttt{\g__tag_mc_parenttree_prop}.)}

Artifacts can have various types like Pagination or Layout. This stored in this variable.
\begin{verbatim}
\tl_new:N \l__tag_mc_artifact_type_tl
\end{verbatim}
\textit{(End of definition for \texttt{\l__tag_mc_parenttree_prop}.)}

This booleans store the stash and artifact status of the mc-chunk.
\begin{verbatim}
\bool_new:N \l__tag_mc_key_stash_bool
\bool_new:N \l__tag_mc_artifact_bool
\end{verbatim}
\textit{(End of definition for \texttt{\l__tag_mc_key_stash_bool} and \texttt{\l__tag_mc_artifact_bool}.)}

Variables used by the keys. \texttt{\l__tag_mc_key_properties_tl} will collect a number of
values. TODO: should this be a pdfdict now?
\begin{verbatim}
\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{verbatim}
\textit{(End of definition for \texttt{\l__tag_mc_key_stash_bool} and others.)}

The commands labels a mc-chunk. It is used if the user explicitly labels the mc-chunk
with the \texttt{label} key. The argument is the value provided by the user. It stores the
attributes \texttt{tagabspage}: the absolute page, \texttt{\g_shipout_readonly_int},
\texttt{tagmcabs}: the absolute mc-counter \texttt{\c@g_@@_MCID_abs_int}. The reference command is
based on \texttt{l3ref}.
\begin{verbatim}
\cs_new:Npn \__tag_mc_handle_mc_label:e #1
{ \__tag_property_record:en{tagpdf-#1}{tagabspage,tagmcabs} }
\end{verbatim}
\textit{(End of definition for \texttt{\__tag_mc_handle_mc_label:e}.)}

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.
\begin{verbatim}
\cs_new_protected:Npn \__tag_mc_set_label_used:n #1 \%#1 labelname
{ \tl_new:c { g__tag_mc_label\tl_to_str:n(#1)_used_tl } }
\end{verbatim}
\textit{(End of definition for \texttt{\__tag_mc_set_label_used:n}.)}

3.2 Functions
These commands 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 used a second time. The argument is a label name set with the `label` key.

TODO: is testing for struct the right test?

```latex
\tag_mc_use:n #1 \{ \_\_tag_whatsits: \}
\cs_set_protected:Npn \tag_mc_use:n #1 \%#1: label name
\begin{verbatim}
\_\_tag_check_if_active_struct:T
\{\
\tl_set:Nn \l__tag_tmpa_tl { \_\_tag_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}
\end{verbatim}
```

(End of definition for `\tag_mc_use:n`. This function is documented on page 68.)

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.

```latex
\tag_mc_artifact_group_begin:n #1 \}
\begin{verbatim}
\_\_tag_mc_end_push:
\_\_tag_mc_begin:n {artifact=#1}
\group_begin:
\_\_tag_stop:n(artifact-group)
\end{verbatim}
```

```latex
\_\_tag_mc_artifact_group_end:
```

```latex
\begin{verbatim}
\_\_tag_end:
\_\_tag_mc_begin_pop:n{}
\end{verbatim}
```

(End of definition for `\tag_mc_artifact_group_begin:n`. This function is documented on page 68.)
This allows to reset the mc-attributes in box. On base and generic mode it should do nothing.

\begin{verbatim}
\cs_new_protected:Npn \tag_mc_reset_box:N #1 {}
\end{verbatim}

(End of definition for \tag_mc_reset_box:N. This function is documented on page 69.)

\begin{verbatim}
\cs_set_protected:Npn \tag_mc_end_push: \tag_mc_begin_pop:n #1 {}
\end{verbatim}

(End of definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 68.)
3.3 Keys

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

the two internal artifact keys are used 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 {} }, __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 69.)

(/shared)
**Part V**

The **tagpdf-mc-generic** module

**Code related to Marked Content (mc-chunks), generic mode**

**Part of the tagpdf package**

1. **Marked content code – generic mode**

   ```latex
   ⟨@=tag⟩
   ⟨@=generic⟩
   \ProvidesExplPackage{tagpdf-mc-code-generic}{2024-02-29}{0.98x}
   {part of tagpdf - code related to marking chunks - generic mode}
   ⟨/generic⟩
   ⟨@=debug⟩
   \ProvidesExplPackage{tagpdf-debug-generic}{2024-02-29}{0.98x}
   {part of tagpdf - debugging code related to marking chunks - generic mode}
   ⟨/debug⟩
   
   **1.1 Variables**
   ⟨@=generic⟩
   \__tag_mc_ref_abspage_tl
   \tl_new:N \__tag_mc_ref_abspage_tl
   (End of definition for \__tag_mc_ref_abspage_tl.)

   \__tag_mc_tmpa_tl
   \tl_new:N \__tag_mc_tmpa_tl
   (End of definition for \__tag_mc_tmpa_tl.)

   \g__tag_mc_marks
   \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.)
   ```
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.

1.2 Functions

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).
\__tag_mc_end_marks: This disables the marks. They can’t be reenabled, so it should only be used in groups.

\__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.

\__tag_mc_store:nnn This inserts the mc-chunk \langle mc-num \rangle into the structure struct-num after the \langle mc-prev \rangle. 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.
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 an 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.
\seq_item:cn { g__tag_mc_#1_marks_seq } \{ 3 \}

\__tag_check_typeout_v:n { =>~ TMB~ not~ missing } \}

\__tag_check_typeout_v:n { =>~ TME~ not~ missing } \}

\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 } 
  } 
\}

(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 and 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:Nnm #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 } 
  } 
\}

79
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 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:Na.)

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 artifically 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
\__tag_tmpa_box
\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:

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!

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).

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

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.
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.)

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.

\bool_if:NTF \g__tag_delayed_shipout_bool
{ 
\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:eV
{ mcid-\int_use:N \c@g__tag_MCID_abs_int }
\__tag_mc_property_mc_clist
\__tag_mc_bdc_shipout:ee
{#1}
{ /MCID-\flag_height:n { __tag/mcid } }
\flag_raise:n { __tag/mcid }- #2
}
}

if the engine is too old, we have to revert to earlier method.
\
\msg_new:nnn { tagpdf } { old-engine }
82
The engine or the PDF management is too old or \ delayed-shipout has been disabled.\n Fast-numbering-of-MC-chunks not available.\n More-compilations will be needed in generic mode.

\msg_warning:nn { tagpdf} { old-engine }
\__tag_prop_new:N \g__tag_MCID_byabspage_prop
\int_new:N \g__tag_MCID_tmp_bypage_int
\cs_generate_variant:Nn \__tag_mc_bdc:nn {ne}

revert the attribute:
\__tag_property_gset:nnnn {tagmcid } { now } {0} { \int_use:N \c@g__tag_MCID_abs_int
\tl_set:Ne \l__tag_mc_ref_abspage_tl
\__tag_property_ref:enn %3 args
\mcid-\int_use:N \c@g__tag_MCID_abs_int
\tagabspage {-1}
\prop_get:NoNTF \g__tag_MCID_byabspage_prop
\l__tag_mc_ref_abspage_tl \l__tag_mc_tmpa_tl
\__tag_property_record:eV \mcid-\int_use:N \c@g__tag_MCID_abs_int\c__tag_property_mc_clist \__tag_mc_bdc:ne

{#1}
\_\_tag\_handle\_stash:n  This is the handler which puts a mc into the the current structure. The argument is the number of the mc. Beside storing the mc into the structure, it also has to record the structure for the parent tree. The name is a bit confusing, it does not handle mc with the stash key . . . . TODO: why does luamode use it for begin + use, but generic mode only for begin?

\_\_tag\_handle\_stash:e

\_\_tag\_mc\_artifact:  Two commands to create artifacts, one without type, and one with. We define also a wrapper handler as luamode will need a different definition. TODO: perhaps later: more properties for artifacts

\_\_tag\_mc\_Artifact:n

\_\_tag\_mc\_Artifact:N

\_\_tag\_struct\_kid\_mc\_gput\_right:nn  { #1}

\_\_tag\_struct\_stack\_current\_tl  { #1}

\prop\_gput:Nee \_\_tag\_mc\_parent\_tree\_prop  { #1}

\_\_tag\_struct\_stack\_current\_tl  

\_\_tag\_mc\_Artifact:  

\_\_tag\_mc\_Artifact:n

\_\_tag\_mc\_Artifact:N
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 }

These are the core public commands to open and close an mc. They don’t need to be in the same group or grouping level, but the code expect that they are issued linearly.
The tag and the state is passed to the end command through a global var and a global boolean.
\cs_set_protected:Npn \tag_mc_begin:n #1 \#1 keyval
\__tag_check_if_active_mc:T
\__tag_check_if_active_mc:TF
\__tag_debug_mc_begin_insert:n { #1 }
\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_set_eq:NN \g__tag_mc_key_tag_tl \tl__tag_struct_tag_tl
\keys_set:nn { __tag / mc } {#1}
\bool_if:NTF \l__tag_mc_artifact_bool
%handle artifact
\__tag_mc_handle_artifact:N \l__tag_mc_artifact_type_tl
\exp_args:NV
\__tag_mc_artifact_begin_marks:oo{\l__tag_mc_key_tl}{\l__tag_mc_key_label_tl}
%handle mcid type
\__tag_check_mc_tag:N \l__tag_mc_mcid_tl
\__tag_mc_handle_mcid:VV
\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}
\exp_args:NV
\__tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl
\bool_if:FN \l__tag_mc_key_stash_bool
{\exp_args:NV\__tag_struct_get_parentrole:nNN
 \l__tag_struct_stack_current_tl
 \l__tag_get_parent_tmpa_tl
 \l__tag_get_parent_tmpb_tl
 \__tag_check_parent_child:VVnnN
 \l__tag_get_parent_tmpa_tl
 \l__tag_get_parent_tmpb_tl
 {MC}\}
 \l__tag_parent_child_check_tl
 \int_compare:nNnT {\l__tag_parent_child_check_tl}<0
 {\prop_get:cnN { g__tag_struct_ \g__tag_struct_stack_current_tl _prop}
 {S}
 \l__tag_tmpa_tl
 \msg_warning:nneee
 { tag }
 {role-parent-child}
 { \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl }
 { MC-(real content) }
 { not-allowed-
 (struct\g__tag_struct_stack_current_tl,-\l__tag_tmpa_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
 }
 ⟨/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:
1.4 Keys

Definitions are different in luamode. \texttt{tag} and \texttt{raw} are expanded as \texttt{\lua_now:e} in lua

\begin{verbatim}
\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
\{     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
\{     \str_set_convert:Noon
\{     \tl_set:Ne \l__tag_mc_key_properties_tl { /Alt< }
\tl_set:Ne \l__tag_mc_key_properties_tl { \l__tag_meta_str>~ }
\},
\alttext .meta:n = {alt=#1},
\actualtext .code:n = % ActualText property
\{     \tl_if_empty:oF{#1}
\{     \str_set_convert:Noon
\{     \tl_set:Ne \l__tag_mc_key_tag_tl { #1 }
\{     \str_set_convert:Noon
\{     \tl_set:Ne \l__tag_mc_key_properties_tl { /ActualText< }
\tl_set:Ne \l__tag_mc_key_properties_tl { \l__tag_meta_str>~ }
\}
\}
\end{verbatim}

(End of definition for \texttt{\tag_mc_begin:n} and \texttt{\tag_mc_end:}. These functions are documented on page 68.)
\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}
}
\langle/generic\rangle

(End of definition for tag (mc-key) and others. These functions are documented on page 69.)
Part VI

The tagpdf-mc-luacode module

Code related to Marked Content (mc-chunks), luamode-specific

Part of the tagpdf package

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

1 Marked content code – luamode code

luamode uses attributes to mark mc-chunks. The two attributes used are defined in the backend file. The backend also load the lua file, as it can contain functions needed elsewhere. The attributes for mc are global (between 0.6 and 0.81 they were local but this was reverted). The attributes are setup only in lua, and one should use the lua functions to set and get them.

\texttt{g\_\texttt{\_\_mc\_type\_attr}}: the value represent the type
\texttt{g\_\texttt{\_\_mc\_cnt\_attr}}: will hold the \texttt{\_\_MCID\_abs\_int} value

Handling attribute needs a different system to number the page wise mcid’s:

\texttt{a \texttt{\_\texttt{tagmc\texttt{\_\begin{...}}}} \texttt{\texttt{\_\texttt{tagmc\texttt{\_\end{}}}}} 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 \texttt{struct \rightarrow mc\_cnt} (in \texttt{\_\_mc\_parent\_tree\_prop} and/or a lua table and at shipout \texttt{mc\_cnt\_\rightarrow \{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:

\texttt{tag}: the type (a string)
\texttt{raw}: more properties (string)
\texttt{label}: a string.
\texttt{artifact}: the presence indicates an artifact, the value (string) is the type.
\texttt{kids}: an array of tables

\{1=\{\texttt{kid}=num2, \texttt{page}=pagenum1\}, 2=\{\texttt{kid}=num2, \texttt{page}=pagenum2\},...,\},

this describes the chunks the mc has been split to by the traversing code

\texttt{parent}: the number of the structure it is in. Needed to build the parent tree.
The main function which wanders through the shipout box to inject the literals. if the new callback is there, it is used.
1.1 Commands

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 {} 

(End of definition for \_tag_add_missing_mcs_to_stream:Nn.)

\__tag_mc_if_in_p: \__tag_mc_if_in:TF \tag_mc_if_in:TF \prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF} 
\int_compare:nNnTF { -2147483647 } = \{ \lua_now:e \{ \int_use:N \c_document_cctab,tex.getattribute(luatexbase.attributes.g__tag_mc_type_attr) \} \} \prg_return_false: } \prg_return_true: \}

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

(End of definition for \__tag_mc_if_in:TF and \tag_mc_if_in:TF. This function is documented on page 68.)

\__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 
\tl_set:Ne\l__tag_tmpa_tl{\lua_now:e{ltx.__tag.func.output_num_from (#1)}} \lua_now:e 
\{ tex.setattribute 
\{ "global", lutexbase.attributes.g__tag_mc_type_attr, \_tag_tmpa_tl \} \} \lua_now:e 
\{ tex.setattribute 
\{ "global", lutexbase.attributes.g__tag_mc_cnt_attr, \_tag_get_mc_abs_cnt: \} }

91
\cs_generate_variant:Nn \__tag_mc_lua_set_mc_type_attr:n { o } \cs_new:Nn \__tag_mc_lua_set_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 ) } \cs_new:Nn \__tag_mc_insert_mcid_kids:n \cs_new:Nn \__tag_mc_insert_mcid_single_kids:n \cs_new_protected:Npn \__tag_mc_insert_mcid_kids: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
\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
{ %\__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\”)
 }\keys\_set:nn { __\_tag / \_mc }{ label={}, \#1 }
\%check that a tag or artifact has been used
\__tag\_check\_mc\_tag:N \l\__tag\_mc\_key\_tag\_tl
\%set the attributes:
\__tag\_mc\_lua\_set\_mc\_type\_attr:o { \l\__tag\_mc\_key\_tag\_tl }
\bool\_if:NF \l\__tag\_mc\_artifact\_bool
{ %store the absolute num name in a label:
\tl_if_empty:NF \l__tag_mc_key_label_tl
\{\exp_args:NV \__tag_mc_handle_mc_label:e \l__tag_mc_key_label_tl\}
\%
if not stashed record the absolute number
\bool_if:NF \l__tag_mc_key_stash_bool
\{\exp_args:NV \__tag_struct_get_parentrole:nNN \g__tag_struct_stack_current_tl \l__tag_get_parent_tmpa_tl \l__tag_get_parent_tmpb_tl \__tag_check_parent_child:VVnnN \l__tag_get_parent_tmpa_tl \l__tag_get_parent_tmpb_tl (MC){}\l__tag_parent_child_check_tl \int_compare:nNnT \l__tag_parent_child_check_tl < {0}
\{\prop_get:cnN \g__tag_struct_ \g__tag_struct_stack_current_tl _prop}{S}\l__tag_tmpa_tl \msg_warning:nneee
\{ tag \}
\{role-parent-child\}
\{ \l__tag_get_parent_tmpa_tl/\l__tag_get_parent_tmpb_tl \}
\{ MC-(real content) \}
\{ not-allowed-\}
\{ struct-\g__tag_struct_stack_current_tl, -\l__tag_tmpa_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 68.)

\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_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 \{ \}
\}
\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)
  \}
\}

\__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 }

1.2 Key definitions

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
      \}
      \{
        \str_set_convert:Noon
        \l__tag_tmpa_str
        \{
          \#1
        \}
        \{
          utf16/hex
        \}
      \}
    \}
  }
\}
\tl_set:Nn \l__tag_mc_key_properties_tl { \l__tag_tmpa_str-> }
\lua_now:e
{
  ltx.__tag.func.store_mc_data
  ( \__tag_get_mc_abs_cnt:;"alt","/Alt~<\str_use:N \l__tag_tmpa_str>"
  )
}
},
alttext .meta:n = {alt=#1},
aactualtext .code:n = % Alt property
{ \tl_if_empty:oF{#1}
  { \str_set_convert:No\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 }
\endluamode

(End of definition for tag (mc-key) and others. These functions are documented on page 69.)
Part VII

The \texttt{tagpdf-struct} module

Commands to create the structure

Part of the \texttt{tagpdf} package

1 Public Commands

\begin{verbatim}
\tag_struct_begin:n{⟨key-values⟩}
\tag_struct_end:
\tag_struct_end:n{⟨tag⟩}
\end{verbatim}

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

\begin{verbatim}
\tag_struct_use:n{⟨label⟩}
\tag_struct_use_num:n{⟨structure number⟩}
\end{verbatim}

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.

\begin{verbatim}
\tag_struct_object_ref:n{⟨struct number⟩}
\end{verbatim}

This is a small wrapper around \texttt{\pdf_object_ref:n} to retrieve the object reference of the structure with the number \texttt{⟨struct number⟩}. This number can be retrieved and stored for the current structure for example with \texttt{\tag_get:n{⟨struct num⟩}}. 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.

\begin{verbatim}
\tag_struct_insert_annot:nn{⟨object reference⟩}{⟨struct parent number⟩}
\end{verbatim}

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

\begin{verbatim}
\tag_struct_parent_int:
\end{verbatim}

This gives back the next free /StructParent number (assuming that it is together with \texttt{\tag_struct_insert_annot:nn} which will increase the number.
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 \texttt{ref} which updates the \texttt{Ref} key (an array).

\section{Public keys}

\subsection{Keys for the structure commands}

\texttt{tag,} This is required. The value of the key is normally one of the standard types listed in the main \texttt{tagpdf} documentation. It is possible to setup new tags/types. The value can also be of the form \texttt{type/NS}, where \texttt{NS} is the shorthand of a declared name space. Currently the names spaces \texttt{pdf}, \texttt{pdf2}, \texttt{mathml} and \texttt{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.

\texttt{stash,} Normally a new structure inserts itself as a kid into the currently active structure. This key prohibits this. The structure is nevertheless from now on “the current active structure” and parent for following marked content and structures.

\texttt{label,} This key sets a label by which one can refer to the structure. It is e.g. used by \texttt{\tag_{-}struct-use:n} (where a real label is actually not needed as you can only use structures already defined), and by the \texttt{ref} key (which can refer to future structures). Internally the label name will start with \texttt{tagpdfstruct-} and it stores the two attributes \texttt{tagstruct} (the structure number) and \texttt{tagstructobj} (the object reference).

\texttt{parent,} 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 \texttt{\tag_get:n}, but one can also use a label on the parent structure and then use \texttt{\property_ref:n\{tagpdfstruct-label\}\{tagstruct\}} to retrieve it.

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

\texttt{alt,} This key inserts an \texttt{/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.
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.

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

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}.

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 stucked to E).

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

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 file 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 file 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.
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.

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

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},
}

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.

3 Variables

Every structure will have a unique, absolute number. I will use a latex counter for the structure count to have a chance to avoid double structures in align etc.
a sequence to store mapping between the structure number and the object number. We assume that structure numbers are assign consecutively and so the index of the seq can be used. A seq allows easy mapping over the structures.

\__tag_seq_new:N \g__tag_struct_objR_seq

In lua mode we have to test if the kids a null
\tl_const:Nn\c__tag_struct_null_tl {null}

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 absolut mc num, the value the pdf directory.
\__tag_prop_new:N \g__tag_struct_cont_mc_prop

A stack sequence for the structure stack. When a sequence is opened it’s number is put on the stack.
\seq_new:N \g__tag_struct_stack_seq
\seq_gpush:Nn \g__tag_struct_stack_seq {0}

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}}

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.
\tl_new:N \g__tag_struct_stack_current_tl
\tl_gset:Nn \g__tag_struct_stack_current_tl {\int_use:N\c@g__tag_struct_abs_int}
\tl_new:N \l__tag_struct_stack_parent_tmpa_tl

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

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

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

All properties have at least the keys
Type StructTreeRoot or StructElem

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

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.

3.1 Variables used by the keys

Use by the tag key to store the tag and the namespace. The role tag variables will hold
locally rolemapping info needed for the parent-child checks
3.2 Variables used by tagging code of basic elements

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.

\prop_new_linked:N \g__tag_struct_dest_num_prop

(End of definition for \g__tag_struct_dest_num_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.

\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.
\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} \}
}\}

\associated{\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:

\begin{verbatim}
\tl_gset:Nn \g__tag_struct_stack_current_tl {0}
\__tag_pdf_name_e:n \cs_new:Npn \__tag_pdf_name_e:n #1{\pdf_name_from_unicode_e:n{#1}}
\end{verbatim}

\associated{\package}{End of definition for \__tag_pdf_name_e:n.}

4.1 Initialization of the StructTreeRoot

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

\begin{verbatim}
\tl_gset:Nn \g__tag_struct_stack_current_tl {0}
\end{verbatim}

\associated{\package}{End of definition for \_tag_pdf_name_e:n.}
Namespaces are pdf 2.0. If the code moves into the kernel, the setting must be probably delayed.

```latex
\pdf_version_compare:NnF < {2.0}
{\__tag_struct_prop_gput:nne
{0}
{rolemap}
{\{StructTreeRoot\}{pdf}}}
```

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

```latex
\prop_new:c { g__tag_struct_debug_0_prop }
\seq_new:c { g__tag_struct_debug_kids_0_seq }
\prop_gset_eq:cc { g__tag_struct_debug_0_prop }{ g__tag_struct_0_prop }
\prop_gremove:cn { g__tag_struct_debug_0_prop }{Namespaces}
```

(End of definition for g__tag_struct_0_prop and g__tag_struct_kids_0_seq.)

### 4.2 Adding the /ID key

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

```latex
\__tag_struct_get_id:n
```

(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:NNnTF < \{2.0\}

{\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:NNnTF \g\_tag\_role\_NS\_prop \{\#3\} \_l\_tag\_get\_tmpc\_tl

{\_tag\_struct\_prop\_gput:nne

\{ \#1 \}

\{ NS \}

\{ \_l\_tag\_get\_tmpc\_tl \%

\}

\}
\cs\generate\_variant:Nn \_tag\_struct\_set\_tag\_info:nnn \{eVV\}

(End of definition for \_tag\_struct\_set\_tag\_info:nnn.)

\_tag\_struct\_get\_parentrole:nnN

We also need a way to get the tag info needed for parent child check from parent structures.

\cs\new\protected\Npn \_tag\_struct\_get\_parentrole:nnNN \#1 \#2 \#3

\%\#1 struct num, \#2 tlvar for tag , \#3 tlvar for \NS

{\prop\_get:cnNnTF

\{ g\_tag\_struct\_#1\_prop \}

\{ parentrole \}

\_l\_tag\_get\_tmpc\_tl

{\tl\_set:Ne \#2{\exp\_last\_unbraced:NV\use\_i:nn \_l\_tag\_get\_tmpc\_tl}

\tl\_set:Ne \#3{\exp\_last\_unbraced:NV\use\_ii:nn \_l\_tag\_get\_tmpc\_tl}

\}

\{\tl\_clear\:N\#2

\tl\_clear\:N\#3

\}
\cs\generate\_variant:Nn \_tag\_struct\_get\_parentrole:nnNN \{eNN\}

107
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).

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 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!

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

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.
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.

\__tag_struct_fill_kid_key:n

\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_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 \}
  \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_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

\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_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

check if we get null

\tl_set:Nn \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\}
4.5 Output of the object

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

\cs_new_protected:Npn \__tag_struct_get_dict_content:nN #1 #2 %#1: structure num
\tl_clear:N #2
\seq_map_inline:cn { \tl_use:Nn \__tag_struct_fill_kid_key:n }
\c__tag_struct_\int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}_entries_seq
\tl_put_right:Ne
  \prop_if_in:cnT
    { g__tag_struct_#1_prop }
    { ##1 }
  \c_space_tl/##1~
Some keys needs the option to format the key, e.g. add brackets for an array
  \cs_if_exist_use:cTF {__tag_struct_format_##1:e}
  { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } }
  { \prop_item:cn{ g__tag_struct_#1_prop } { ##1 } }
\end{definition}

\__tag_struct_format_Ref:n Ref is an array, we store only the content to be able to extend it so the formatting command adds the brackets:
\cs_new:Nn\__tag_struct_format_Ref:n{[#1]}
\cs_generate_variant:Nn\__tag_struct_format_Ref:n{e}
\end{definition}

\__tag_struct_write_obj:n This writes out the structure object. This is done in the finish code, in the tree module and guarded by the tree boolean.
\cs_new_protected:Npn \__tag_struct_write_obj:n #1 \% #1 is the struct num
{ \pdf_object_if_exist:nTF { __tag/struct/#1 } }
\end{definition}
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 } {P}\l__tag_tmpb_tl
  \prop_gput:cn{ g__tag_struct_#1_prop } {P}{\pdf_object_ref:n { __tag/struct/0 }}
  \prop_gput:cn{ g__tag_struct_#1_prop } {S}/{Artifact}
\seq_if_empty:CF {g__tag_struct_kids_#1_seq}
  \msg_warning:nnee
  {tag}
  {struct-orphan}
\__tag_struct_insert_annot:n

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 }\pdannot_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:n {obj ref}{parent num} \tag_mc_end:\tag_struct_end:

\cs_new_protected:Npn \__tag_struct_insert_annot:n #1 #2 %#1 object reference to the annotation/%#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
\begin{verbatim}
\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
}
{
  \pdf_pageobject_ref:n { \__tag_property_ref:nnn {@tag@objr@page@#2 }{ tagabspage }{1} }
}
% add the parent obj number to the parent tree:
\exp_args:Nne
\__tag_parenttree_add_objr:nn
{ #2 }
{ \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl } }
% increase the int:
\int_gincr:N \c@g__tag_parenttree_obj_int
\end{verbatim}

(End of definition for \__tag_struct_insert_annot:nn.)

\__tag_get_data_struct_tag: this command allows \tag_get:n to get the current structure tag with the keyword structure_tag.
\cs_new:Npn \__tag_get_data_struct_tag:
{ \exp_args:Nne \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 structure_id.
\cs_new:Npn \__tag_get_data_struct_id:
{ \__tag_struct_get_id:n {g__tag_struct_stack_current_tl} }

(End of definition for \__tag_get_data_struct_id:.)
\_\_tag\_get\_data\_struct\_num: this command allows \texttt{\_\_tag\_get\_n} to get the current structure number with the keyword \texttt{struct\_num}. We will need to handle nesting.

\begin{verbatim}
\cs_new:Npn \__tag_get_data_struct_num: 
{ \g__tag_struct_stack_current_tl }
\end{verbatim}

(End of definition for \_\_tag\_get\_data\_struct\_num:.)

\_\_tag\_get\_data\_struct\_counter: this command allows \texttt{\_\_tag\_get\_n} to get the current state of the structure counter with the keyword \texttt{struct\_counter}. By comparing the numbers it can be used to check the number of structure commands in a piece of code.

\begin{verbatim}
\cs_new:Npn \__tag_get_data_struct_counter: 
{ \int_use:N \c@g__tag_struct_abs_int }
\end{verbatim}

(End of definition for \_\_tag\_get\_data\_struct\_counter:.)

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.

\begin{verbatim}
\socket_new:nnn { tag/struct/tag }{1}{ latex-tags }
\socket_new_plug:nnn { tag/struct/tag }{ latex-tags }
{ \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#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 }
{ \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#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:VVNN \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 \tl_set:Ne \l__tag_tmpa_seq \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}
\end{verbatim}

115
\keys_define:nn { __tag / struct }
{
  label .tl_set:N = \l__tag_struct_key_label_tl,
  stash .bool_set:N = \l__tag_struct_elem_stash_bool,
  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},
  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>}
}
},
lang .code:n = % Lang property
{ \__tag_struct_prop_gput:nne
{ \int_use:N \c@g__tag_struct_abs_int }
{ Lang }
{ (#1) }
},
Ref is an array, the brackets are added through the formatting command.
ref .code:n = % ref property
{ \tl_clear:N\l__tag_tmpa_tl
\clist_map_inline:on {#1}
{ \tl_put_right:Ne \l__tag_tmpa_tl
\{\tl_put_right:Ne \__tag_property_ref:en{tagpdfstruct-##1}{tagstructobj} \} }
\__tag_struct_gput_data_ref:ee
{ \int_use:N \c@g__tag_struct_abs_int } {\l__tag_tmpa_tl}
},
E .code:n = % E property
{ \str_set_convert:Nnon
\l__tag_tmpa_str
{ #1 }
{ default }
{ utf16/hex }
\__tag_struct_prop_gput:nne
(End of definition for label (struct-key) and others. These functions are documented on page 99.)

This variable is used to number the AF-object names

\begin{verbatim}
\int_new:N \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}
\l__tag_tmpa_tl \__tag_struct_add_AF:ee
\__tag_struct_prop_gput:nne { \int_use:N \c@g__tag_struct_abs_int } { AF }
\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
\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
\tl_use:c { g__tag_struct\_int_eval:n {\c@g__tag_struct_abs_int}\_AF_tl }
\}
\end{verbatim}

AF\_i(struct-key)
AFref\_i(struct-key)
AFinline\_i(struct-key)
AFinline-\_0,\_o(struct-key)
\tl_gput_right:ce
\{ \c_space_tl \#2 \}
\}
\{ \tl_new:c \}
\tl_gset:ce \{ \c_space_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}
\{ \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 \}{\pdf_object_ref:e}
\{ \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} \}
\}
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 { 0 }\pdf_object_ref:n {#1}}
 \__tag_struct_prop_gput:nne
 { 0 }
 { AF }
 { \tl_use:c
 { g__tag_struct_0_AF_tl }
 }
}
}

(End of definition for root-AF (setup-key). This function is documented on page 101.)
The structure number of the parent is either taken from the stack or has been set with the parent key.

The structure number of the parent is either taken from the stack or has been set with the parent key.
we also store which role to use for parent/child test. If the role is one of Part, Div, NonStruct we have to retrieve it from the parent. If the structure is stashed, this must be updated!

\str_case:VnTF \l__tag_struct_roletag_tl
{\Part} {} 
{\Div} {} 
{\NonStruct} {}
}
{
\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}
}

\seq_gpush:Ne \g__tag_struct_tag_stack_seq
{\l__tag_struct_roletag_tl}{\l__tag_struct_roletag_NS_tl}
\tl_gset:NV \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
%\seq_show:N \g__tag_struct_stack_seq
\bool_if:NF \l__tag_struct_elem_stash_bool
{
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. For now we ignore the
namespace!

\_\_tag\_struct\_get\_parent\_role:eNN
\{\_\_tag\_struct\_stack\_parent\_tmpa\_tl\}
\_\_tag\_get\_parent\_tmpa\_tl
\_\_tag\_get\_parent\_tmpb\_tl
\_\_tag\_check\_parent\_child:VVVVV
\{\_\_tag\_get\_parent\_tmpa\_tl\}
\{\_\_tag\_get\_parent\_tmpb\_tl\}
\g\_tag\_struct\_tag\_tl
\g\_tag\_struct\_tag\_NS\_tl
\_\_tag\_parent\_child\_check\_tl
\int\_compare:nNnT \{\_\_tag\_parent\_child\_check\_tl\}<0
{
  \prop\_get:cnN
  \{ g\_\_tag\_struct\_ \_\_tag\_struct\_stack\_parent\_tmpa\_tl \_\_\_prop\}
  \{S\}
  \_\_tag\_tmpa\_tl
  \msg\_warning:nnee
  \{ tag \}
  \{role\_parent\_child\}
  \{ \_\_tag\_get\_parent\_tmpa\_tl/\_\_tag\_get\_parent\_tmpb\_tl \}
  \{ g\_tag\_struct\_tag\_tl/g\_tag\_struct\_tag\_NS\_tl \}
  \{ not\_allowed- \}
  \{ \_\_tag\_struct\_stack\_parent\_tmpa\_tl,-/\_\_tag\_tmpa\_tl\}
  \c\_space\_tl-->/\struct\-_\int\_eval:n \{/c@g\_\_tag\_struct\_abs\_int\}
\}
\cs\_set\_eq:NW \_\_tag\_role\_remap\_tag\_tl \g\_tag\_struct\_tag\_tl
\cs\_set\_eq:NW \_\_tag\_role\_remap\_NS\_tl \g\_tag\_struct\_tag\_NS\_tl
\_\_tag\_role\_remap:
\cs\_get\_eq:NW \g\_tag\_struct\_tag\_tl \_\_tag\_role\_remap\_tag\_tl
\cs\_get\_eq:NW \g\_tag\_struct\_tag\_NS\_tl \_\_tag\_role\_remap\_NS\_tl
\_\_tag\_struct\_set\_tag\_info:eVVV
\{ \int\_use:N \c@g\_\_tag\_struct\_abs\_int \}
\g\_tag\_struct\_tag\_tl
\g\_tag\_struct\_tag\_NS\_tl
}

Set the Parent.

\_\_tag\_struct\_prop\_gput:nne
\{ \int\_use:N \c@g\_\_tag\_struct\_abs\_int \}
\{ P \}
\{
  \pdf\_object\_ref:e \{ \_\_tag\_struct/\_\_tag\_struct\_stack\_parent\_tmpa\_tl \}
\}
\%record this structure as kid:
\%\tl\_show:N \g\_\_tag\_struct\_stack\_current\_tl
\%\tl\_show:N \_\_tag\_struct\_stack\_parent\_tmpa\_tl
\_\_tag\_struct\_kid\_struct\_gput\_right:e
\{ \_\_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_/\_\_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.

\prop_gset_eq:cc
\prop_gput:cne
\bool_if:NTF \l__tag_struct_elem_stash_bool
\prop_gput:cne
\__tag_debug_struct_begin_insert:n { #1 }
\group_end:
\__tag_debug_struct_begin_ignore:n { #1 }
\cs_set_protected:Nn \tag_struct_end:
\__tag_check_if_active_struct:T
\__tag_check_no_open_struct: }
\seq_get:NNT \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
\tl_gset:NV \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
\tag_struct_use:n  
This command allows to use a stashed structure in another place. TODO: decide how it should be guarded. Probably by the struct-check.

\cs_new_protected:Npn \tag_struct_use:n #1 {}
\prop_if_exist:cTF
{ g__tag_struct__tagpdfstruct-#1{tagstruct}{unknown}_prop }
{ %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 }
  { \__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0} }
%add the current structure to the labeled one as parents
\__tag_prop_gput:cne
{ g__tag_struct__tagpdfstruct-#1{tagstruct}{0}_prop }
{ P }
{ \pdf_object_ref:e { __tag/struct/\g__tag_struct_stack_current_tl } }

debug code
\prop_gput:cne
{ g__tag_struct__tagpdfstruct-#1{tagstruct}{0}_prop }
{ P }
{ }
\prop_gput:cne
{ g__tag_struct__tagpdfstruct-#1{tagstruct}{0}_prop }
{ P }
{ parent-structure:\g__tag_struct_stack_current_tl\c_space_tl=}
\prop_gput:cne
{ g__tag_struct_stack_current_tl }
\prop_gput:cne
{ g__tag_struct_tag_tl }
\prop_gput:cne
{ g__tag_struct_tag_tl }

(End of definition for \tag_struct_begin:n and \tag_struct_end:. These functions are documented on page 98.)
check if the tag is allowed as child. Here we have to retrieve the tag info for the child, while the data for the parent is in the global tl-vars:

\tag_struct_get_parentrole:eNN
\__tag_property_ref:enn{tagpdfstruct-#1}{tagstruct}{0}
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\__tag_check_parent_child:VVVVN
\g__tag_struct_tag_tl
\g__tag_struct_tag_NS_tl
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\l__tag_parent_child_check_tl
\int_compare:nNnT \l__tag_parent_child_check_tl<0
{
\cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
\cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
\__tag_role_remap:
\cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
\cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
\__tag_struct_set_tag_info:eVV
\int_compare:nNnT {\l__tag_parent_child_check_tl}<0
{
\cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
\cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
\__tag_struct_set_tag_info:eVV
\int_compare:nNnT {\l__tag_parent_child_check_tl}<0
{
\cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
\cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
\__tag_struct_set_tag_info:eVV
\msg_warning:nnn{ tag }{struct-label-unknown}{#1}
}
}
\msg_warning:nnn{ tag }{struct-label-unknown}{#1}
}
check if the tag is allowed as child. Here we have to retrieve the tag info for the child, while the data for the parent is in the global tl-vars:

\__tag_struct_get_parentrole:eNN
{#1}
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\__tag_check_parent_child:VVVVN
\g__tag_struct_tag_tl
\g__tag_struct_tag_NS_tl
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\l__tag_parent_child_check_tl
\int_compare:nNnT {\l__tag_parent_child_check_tl}<0
{
  \cs_set_eq:NN \l__tag_role_remap_tag_tl \g__tag_struct_tag_tl
  \cs_set_eq:NN \l__tag_role_remap_NS_tl \g__tag_struct_tag_NS_tl
  \__tag_role_remap:
  \cs_gset_eq:NN \g__tag_struct_tag_tl \l__tag_role_remap_tag_tl
  \cs_gset_eq:NN \g__tag_struct_tag_NS_tl \l__tag_role_remap_NS_tl
  \__tag_struct_set_tag_info:eVV
\}
\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 98.)

\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.
\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 only keyword is ref which updates the Ref key (an array)

\tag_struct_gput:nn
\tag_struct_gput:ee
\tag_struct_gput: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.
7 Attributes and attribute classes

7.1 Variables
\g\_\_\_tag\_attr\_entries\_prop will store attribute names and their dictionary content.
\g\_\_\_tag\_attr\_class\_used\_seq will hold the attributes which have been used as class name. \l\_\_\_tag\_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\_\_\_tag\_attr\_objref\_prop

7.2 Commands and keys
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.

\tagpdfsetup
{role/new-attribute =
 {TH-col}{/O /Table /Scope /Column},
role/new-attribute =
 {TH-row}{/O /Table /Scope /Row},
}
attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.

attribute-class\textunderscore uni2423(struct-key) attribute-class has to store the used attribute names so that they can be added to the ClassMap later.
attribute\textsubscript{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_if_in:NnF \g__tag_attr_entries_prop {##1}
{ \msg_error:nnn { tag } { attr-unknown } { ##1 } }
} \prop_if_in:NnF \g__tag_attr_objref_prop {##1}
{ \prop_item:Nn \g__tag_attr_entries_prop {##1} \prop_item:Nn \g__tag_attr_objref_prop {##1}\pdf_object_unnamed_write:ne
{ \dict } }
\prop_gput:Ne \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_put_right:Ne \l__tag_attr_value_tl
{ \int_compare:nT { \seq_count:N \l__tag_tmpa_seq > 1 }{}%}
End of definition for attribute (struct-key). This function is documented on page 101.
Part VIII
The tagpdf-luatex.def
Driver for luatex
Part of the tagpdf package

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

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

%Hm not quite sure about the naming
\cs_set:Npn \__tag_seq_item:cn #1 #2
{ \lua_now:e { tex.print(ltx.__tag.tables.#1[#2]) }
}
\cs_set:Npn \__tag_prop_item:cn #1 #2
{ \lua_now:e { tex.print(ltx.__tag.tables.#1["#2"] ) }
}

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

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

\end{verbatim}

The module declaration

\begin{verbatim}
(*lua
-- tagpdf.lua
-- Ulrike Fischer

local ProvidesLuaModule = {
  name = "tagpdf",
  version = "0.98x", --TAGVERSION
  date = "2024-02-29", --TAGDATE
  description = "tagpdf lua code",
  license = "The LATEX Project Public License 1.3c"
}

134
if luatexbase and luatexbase.provides_module then
  luatexbase.provides_module (ProvidesLuaModule)
end

--{}

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

Some comments about the lua structure.
--{}

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

ltx.__tag.mc will contain mc connected data.
ltx.__tag.struct will contain structure related data.
ltx.__tag.page will contain page data
ltx.__tag.tables contains also data from mc and struct (from older code). This needs cleaning
There are certainly dublettes, but I don’t dare yet ...
ltx.__tag.func will contain (public) functions.
ltx.__tag.trace will contain tracing/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_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,name,mctypeprev) : the main
ltx.__tag.func.mark_shipout (): a wrapper around the core function which inserts the last EMC
ltx.__tag.func.fill_parent_tree_line (page): outputs the entries of the parenttree for this page
ltx.__tag.func.output_parenttree(): outputs the content of the parenttree
ltx.__tag.func.pdf_object_ref(name): outputs the object reference for the object name
ltx.__tag.func.markspaceon(), ltx.__tag.func.markspaceoff(): (de)activates the marking of possible
ltx.__tag.trace.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is the current log level
ltx.__tag.trace.show_all_mc_data (max,loglevel): shows a maximum about mc’s if the current log level
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.showspace : boolean
--{]
This set-ups the main attribute registers. The mc_type attribute stores the type (P, Span etc) encoded as a num, The mc_cnt attribute stores the absolute number and allows so to see if a node belongs to the same mc-chunk.

The interwordspace attr is set by the function \texttt{\_\_mark\_spaces}, and marks the place where spaces should be inserted. The interwordfont attr is set by the function \texttt{\_\_mark\_spaces} too and stores the font, so that we can decide which font to use for the real space char.

\begin{verbatim}
local mctypeattributeid = luatexbase.new_attribute("g__tag_mc_type_attr")
local mccntattributeid = luatexbase.new_attribute("g__tag_mc_cnt_attr")
local iwspaceattributeid = luatexbase.new_attribute("g__tag_interwordspace_attr")
local iwfontattributeid = luatexbase.new_attribute("g__tag_interwordfont_attr")
\end{verbatim}

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

\begin{verbatim}
local tagunmarkedbool = token.create("g__tag_tagunmarked_bool")
local truebool = token.create("c_true_bool")
\end{verbatim}

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

\begin{verbatim}
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 nodelast = node.tail
local nodeslide = node.slide
local noderemove = node.remove
local nodeinsertafter = node.insert_after
local nodeinsertbefore = node.insert_before
local pdfpageref = pdf.pageref
\end{verbatim}

\begin{verbatim}
local fonthashes = fonts.hashes
local identifiers = fonthashes.identifiers
local fontid = font.id
\end{verbatim}

\begin{verbatim}
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")
\end{verbatim}

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

\begin{verbatim}
ltx = ltx or { }
\end{verbatim}
2 Logging functions

__tag_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 .. not found",1)
  end
end

(End of definition for ltx.__tag.trace.show_seq.)

__tag_pairs_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
    return a[i]
  end
end

(End of definition for ltx.__tag.trace.show_prop.)
if a[i] == nil then return nil
else return a[i], prop[a[i]]
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

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" .. 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 ",.. kid .. " on page ". .. v.page,loglevel)
    end
  end
else
  __tag_log ("mc" .. num .. " not found",loglevel)
end
end

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

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 ("mc" .. num .. " not found",loglevel)
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

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.

ltx.__tag.trace.show_struct_data
This function shows some struct data. Unused but kept for debugging.
249  __tag_log ("struct ".num."":"..tostring(k).."=>"..tostring(v),1)
250  end
251  else
252  __tag_log ("struct ".num." not found ",1)
253  end
254  end

(End of definition for ltx.__tag.trace.show_struct_data.)

3 Helper functions

3.1 Retrieve data functions

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

255  local __tag_get_mc_cnt_type_tag = function (n)
256  local mc_cnt = nodegetattribute(n,mccntattributeid) or -1
257  local mc_type = nodegetattribute(n,mctypeattributeid) or -1
258  local tag = ltx.__tag.func.get_tag_from(mctype)
259  return mc_cnt,mc_type,tag
260  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.

261  local function __tag_get_mathsubtype (mathnode)
262  if mathnode.subtype == 0 then
263  subtype = "beginmath"
264  else
265  subtype = "endmath"
266  end
267  return subtype
268  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.

269  ltx.__tag.tables.role_tag_attribute = {}
270  ltx.__tag.tables.role_attribute_tag = {}

(End of definition for ltx.__tag.tables.role_tag_attribute.)

ltx.__tag.func.alloctag

271  local __tag_alloctag =
272  function (tag)
273  if not ltx.__tag.tables.role_tag_attribute[tag] then
274    table.insert(ltx.__tag.tables.role_attribute_tag,tag)
275    ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
276  __tag_log ("Add ".tag." ",ltx.__tag.tables.role_tag_attribute[tag],3)
277  end
278  end
279  ltx.__tag.func.alloctag = __tag_alloctag

139
These functions take as argument a string \texttt{tag}, and return the number under which it is recorded (and so the attribute value). The first function outputs the number for lua, while the output function outputs to tex.

```lua
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
```

(End of definition for \texttt{ltx.__tag.func.alloctag}.)

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

```lua
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
```

(End of definition for \texttt{__tag_get_num_from}, \texttt{ltx.__tag.func.get_num_from}, and \texttt{ltx.__tag.func.output_num_from}.)

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

```lua
function ltx.__tag.func.store_mc_data (num,key,data)
    ltx.__tag.mc[num][key] = ltx.__tag.mc[num] or { }
end
```

(End of definition for \texttt{__tag_get_tag_from}, \texttt{ltx.__tag.func.get_tag_from}, and \texttt{ltx.__tag.func.output_tag_from}.)
3.2 Functions to insert the pdf literals

3.2 Functions to insert the pdf literals

ltx.__tag.mc[num][key] = data
ltx.__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)
ltx.__tag.trace.log("INFO TAG-STORE-MC-KID: "..mcnum.. " => " .. kid.. " on page " .. page,3)
ltx.__tag.mc[mcnum]["kids"] = ltx.__tag.mc[mcnum]["kids"] or { }
local kidtable = {kid=kid,page=page}
tableinsert(ltx.__tag.mc[mcnum]["kids"], kidtable )
end
(End 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
ltx.__tag.trace.log ("INFO MC-KID-NUMBERS: ".. mcnum .. "has " .. num .. "KIDS",4)
return num
end
(End of definition for ltx.__tag.func.mc_num_of_kids.)

__tag_backend_create_emc_node __tag_backend_create_emc_node
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 /ENC pdfmark end"
return emcnode
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

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.)

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
    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.)

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

ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref

local function __tag_pdf_object_ref (name)
  local tokenname = 'c__pdf_backend_object_'..name..'_int'
  local object = token.create(tokenname).mode ..' 0 R'
  return object
end

ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref

4 Function for the real space chars

__tag_show_spacemark

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

ltx.__tag.func.pdf_object_ref=__tag_pdf_object_ref

4 Function for the real space chars

__tag_show_spacemark

A debugging function, it is used to inserts red color markers in the places where space
chars can go, it can have side effects so not always reliable, but ok.
if tex.outputmode == 0 then
    -- ignore dvi mode for now
else
    pdfstring = node.new("whatsit","pdf literal")
    pdfstring.data = string.format("q %s RG %s rg 0.4 w 0 %g m 0 %g l S Q",-
            markcolor,markcolor,-3,markheight)
    head = node.insert_after(head,current,pdfstring)
end

return head
end

(End of definition for __tag_show_spacemark.)

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.)

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 then
                nodesetattribute(glyph.next,iwspaceattributeid,1)
                nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
            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
    end
end

ltx.__tag.func.showspaces = __tag_mark_spaces

(End of definition for __tag_mark_spaces and ltx.__tag.func.showspaces.)

--[[ 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 then
                nodesetattribute(glyph.next,iwspaceattributeid,1)
                nodesetattribute(glyph.next,iwfontattributeid,glyph.font)
            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
    end
end
-- look also back
if glyph.prev and (glyph.prev.id == GLUE)
    and not inside_math
    and (glyph.prev.width >0)
    and not nodehasattribute(glyph.prev,iwspaceattributeid)
then
    nodesetattribute(glyph.prev,iwspaceattributeid,1)
    nodesetattribute(glyph.prev,iwfontattributeid,glyph.font)
-- for debugging
if ltx.__tag.trace.showspaces then
    __tag_show_spacemark (head,glyph)
end
end
elseif id == PENALTY then
    local glyph = n
    -- ltx.__tag.trace.log ("PENALTY ".nsubtype."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.)

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

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

ltx.__tag.func.markspaceon=__tag_activate_mark_space

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

ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space
We need two local variables to setup a default space char.

```lisp
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:

```lisp
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
```

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.

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

\texttt{ltx.__tag.func.mc_insert_kids} \\
This is the main function to insert the K entry into a StructElem object. It is used in \texttt{tagpdf-mc-luacode} module. The \texttt{single} attribute allows to handle the case that a single \texttt{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.

\texttt{ltx.__tag.func.store_struct_mcabs} \\
This function is used in the tagpdf-mc-luacode module. It store the absolute count of the \texttt{mc} into the current structure. This must be done ordered.
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 one mc chunk, the content should be ordered
tableinsert(ltx.__tag.struct[structnum]["mc"],mcnum)
ltx.__tag.trace.log("INFO TEX-MC-INTO-STRUCT: ".
mcnum.." inserted in struct ".structnum,3)
-- but every mc can only be in one structure
ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
ltx.__tag.mc[mcnum]["parent"] = structnum
end

(End 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
ltx.__tag.trace.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)
nodel.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,
cmcmcnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for
mcmcntprev: num, the attribute cnt of the previous node/whatever - if different we have a change
mcopen: num, records if some bdc/emc is open
These arguments are only needed for log messages, if not present are replaces by fix strings.
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. (???)
local head = box.head -- ShipoutBox is a vlist?
if head then
  mccnthead, mctypehead, taghead = __tag_get_mc_cnt_type_tag (head)
ltx.__tag.trace.log ("INFO TAG-HEAD: " .. node.type(node.getid(head)) ..
  " MC" .. tostring(mccnthead) ..
  " => TAG " .. tostring(mctypehead) ..
  " => " .. tostring(taghead), 3)
else
  ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is ".
tostring(head), 3)
end

for n in node.traverse(head) do
  local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
  node.spaceattr = node.getattribute(n,iwspaceattributeid) or -1
  ltx.__tag.trace.log ("INFO TAG-NODE: " .. node.type(node.getid(n)) ..
    " MC" .. tostring(mccnt) ..
    " => TAG " .. tostring(mctype) ..
    " => " .. tostring(tag), 3)
  if n.id == HLIST then -- enter the hlist
    mcopen, mcpagecnt, mccntprev, mctypeprev=
    ltx.__tag.func.mark_page_elements (n, mcpagecnt, mccntprev, mcopen, "INTERNAL HLIST", mctypeprev)
  elseif n.id == VLIST then -- enter the vlist
    mcopen, mcpagecnt, mccntprev, mctypeprev=
    ltx.__tag.func.mark_page_elements (n, mcpagecnt, mccntprev, mcopen, "INTERNAL VLIST", mctypeprev)
  elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but this
    -- has been done if the previous shipout wandering, so here it
    -- is ignored
  elseif n.id == LOCAL_PAR then -- local_par is ignored
  elseif n.id == PENALTY then -- penalty is ignored
  elseif n.id == KERN then -- kern is ignored
    ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: " .. node.type(node.getid(n)) .. " ..n.subtype, 4)
else

end

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

-- math is currently only logged.
-- we could mark the whole as math
for inner processing the mlist_to_hlist callback is probably needed.
if n.id == MATH then
  ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: ".
    node.type(node.getid(n)).." ".__tag_get_mathsbsytype(n),4)
end
endmath
ltx.__tag.trace.log("INFO TAG-MC-COMpare: current ".
  mccnt.." prev ".mccntprev,4)
if mccnt==mccntprev then -- a new mc chunk
  ltx.__tag.trace.log ("INFO TAG-NEW-MC-NODE: ".
    node.type(node.getid(n))..
    " MC"..tostring(mccnt).
    " <=> PREVIOUS "..tostring(mccntprev),4)
if mcopen~=0 then -- there is a chunk open, close it (hope there is only one ... 
  box.list=__.tag_insert_emc_node (box.list,n)
  mcopen = mcopen - 1
  ltx.__tag.trace.log ("INFO TAG-INSERT-EMC: ".
    mcpagecnt .. " MCOPEN = " .. mcopen,3)
if mcopen ==0 then
  ltx.__tag.trace.log ("WARN TAG-OPEN-MC: ". mcopen,1)
end
end
if ltx.__tag.mc[mccnt] then
  if ltx.__tag.mc[mccnt]["artifact"] then
    ltx.__tag.trace.log("INFO TAG-INSERT-ARTIFACT: ".
      tostring(ltx.__tag.mc[mccnt]["artifact"],3)
    if ltx.__tag.mc[mccnt]["artifact"] == "" then
      box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
    else
      box.list = __tag_insert_bdc_node (box.list,n,"Artifact", "/Type /"..ltx.__tag.mc[mccnt]"
    end
  else
    ltx.__tag.trace.log("INFO TAG-INSERT-TAG: ".
      tostring(tag),3)
    mcpagecnt = mcpagecnt +1
    ltx.__tag.trace.log ("INFO TAG-INSERT-BDC: ".mcpagecnt,3)
    local dict= "/MCID ..ltx.__tag.mc[mccnt]["actualtext"] then
  if ltx.__tag.mc[mccnt]["actualtext"] then
    ltx.__tag.trace.log("INFO TAG-USE-ALt: ".
      tostring(ltx.__tag.mc[mccnt]["actualtext"],3)
    dict= dict .. " ".ltx.__tag.mc[mccnt]["actualtext"]
    end
end
if ltx.__tag.mc[mccnt]["actualtext"] then
  ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: ".
    tostring(ltx.__tag.mc[mccnt]["actualtext"],3)
    dict= dict .. " ".ltx.__tag.mc[mccnt]["actualtext"]
end
150
box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
ltx.__tag.func.store_mc_kid (mccnt,mcpagenr,abspage)
ltx.__tag.func.store_mc_in_page(mccnt,mcpagenr,abspage)
ltx.__tag.trace.show_mc_data (mccnt,3)
end
mcopen = mcopen + 1
else
  if tagunmarkedbool.mode == truebool.mode then
    ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
    box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
    mcopen = mcopen + 1
  else
    ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
  end
end
mccntprev = mccnt
end
end -- end if
end -- end for
if head then
  mccnthead, mctypehead,taghead = __tag_get_mc_cnt_type_tag (head)
ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " ..
node.type(node.getid(head)) ..
  " MC" .. tostring(mccnthead) ..
  " => TAG " .. tostring(mctypehead) ..
  " => " .. tostring(taghead),4)
else
  ltx.__tag.trace.log ("INFO TAG-ENDHEAD: " .. tostring(head),4)
end
ltx.__tag.trace.log ("INFO TAG-QUITTING-BOX " ..
tostring(name) ..
  " TYPE " .. node.type(node.getid(box)),4)
return mcopen,mcpagenr,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 emcnodenode = __tag_backend_create_emc_node ()
    local list = box.list
    if list then
      list = node.insert_after (list,node.tail(list),emcnodenode)
      mcopen = mcopen - 1
      ltx.__tag.trace.log ("INFO SHIPOUT-INSERT-LAST-EMC: MCOPEN " .. mcopen,3)
    else
      ltx.__tag.trace.log ("WARN SHIPOUT-UPS: this shouldn’t happen",0)
    end
  end
end
ltx.__tag.trace.log("WARN SHIPOUT-MC-OPEN: " .. mcopen,1)
end
end
end

(End of definition for ltx.__tag.func.mark_shipout.)

## 6 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.

```lua
function ltx.__tag.func.fill_parent_tree_line (page)
    -- we need to get page->i=kid->mcnum->structnum
    -- pay attention: the kid numbers and the page number in the parent tree start with 0!
    local numsentry ="
    local pdfpage = page-1
    if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
        mcchunks=#ltx.__tag.page[page]
        ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: " ..
            for i=0,mcchunks do
                ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: ",
            numsentry = pdfpage .. " [" .. objref .. "]
        ltx.__tag.trace.log("INFO PARENTTREE-NO-DATA: page ",
            end
            else
                numsentry = pdfpage .. " [" .. objref .. " ]
            ltx.__tag.trace.log("INFO PARENTTREE-NO-DATA: page ",
                end
                else
                    ltx.__tag.trace.log("INFO PARENTTREE-NO-DATA: page ",
```
numsentry = pdffile.." []"
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.)

(/lua)
Part IX

The tagpdf-roles module

Tags, roles and namespace code

Part of the tagpdf package

The `add-new-tag` key can be used in \texttt{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 \texttt{tagstructbegin}.

- **namespace**: This is the namespace of the new tag. The value should be a shorthand of a namespace. The allowed values are currently \texttt{pdf}, \texttt{pdf2}, \texttt{mathml}, \texttt{latex}, \texttt{latex-book} and \texttt{user}. The default value (and recommended value for a new tag) is \texttt{user}. The public name of the user namespace is \texttt{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 \texttt{pdf} set, in PDF 2.0 from the \texttt{pdf}, \texttt{pdf2} and \texttt{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.

\texttt{\tag_check_child:nn}\{⟨tag⟩\}{⟨namespace⟩}\{⟨true code⟩\}\{⟨false code⟩\}

This checks if the tag \texttt{⟨tag⟩} from the name space \texttt{⟨namespace⟩} can be used at the current position. In tagpdf-base it is always true.

\texttt{\ProvidesExplPackage {tagpdf-roles-code} {2024-02-29} {0.98x}}

\texttt{\{part of tagpdf - code related to roles and structure names\}}

\texttt{\{/header\}}

1 Code related to roles and structure names


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.

\l__tag_role_debug_prop This property is used to pass some info around for info messages or debugging.
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.

```latex
\prop_new:N \g__tag_role_tags_NS_prop
```

(End of definition for \g__tag_role_tags_NS_prop.)

With pdf 2.0 we store the class in the NS dependant 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.

```latex
\prop_new:N \g__tag_role_tags_class_prop
```

(End of definition for \g__tag_role_tags_class_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.

```latex
\g__tag_role_NS_prop
```

mathml \url{http://www.w3.org/1998/Math/MathML}

pdf2 \url{http://iso.org/pdf2/ssn}

pdf \url{http://iso.org/pdf/ssn} (default)

user \c__tag_role_userNS_id_str (random id, for user tags)

latex \url{https://www.latex-project.org/ns/dflt/2022}

latex-book \url{https://www.latex-project.org/ns/book/2022}

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.

```latex
\prop_new:N \g__tag_role_NS_prop
```

(End of definition for \g__tag_role_NS_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.

```latex
\prop_new:N \g__tag_role_index_prop
```

(End of definition for \g__tag_role_index_prop.)

This variable is used to pass more infos to debug messages.

```latex
\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.

```latex
\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
```

(End of definition for \l__tag_role_tag_tmpa_tl and others.)
1.2 Namespaces

The following commands setup a name space. With pdf version $\langle 2.0$ 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 . . . .

This is the object which contains the normal RoleMap. It is probably not needed in pdf $2.0$ but currently kept.

\begin{verbatim}
\pdfdict_new:n \g__tag_role/RoleMap_dict
\prop_new:N \g__tag_role_rolemap_prop

(End of definition for \g__tag_role/RoleMap_dict and \g__tag_role_rolemap_prop.)
\end{verbatim}

\begin{verbatim}
\__tag_role_NS_new:nnn\{\langle shorthand\}\{\langle 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}
{\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}
{\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\}}
{\prop_gput:Nne \g__tag_role_NS_prop \{#1\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_prop \{#2\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_class_prop \{#3\}{}}
{\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\}}
{\prop_gput:Nne \g__tag_role_NS_prop \{#1\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_prop \{#2\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_class_prop \{#3\}{}}
{\pdf_object_new:n \{tag/NS/#1\}}
{\pdfdict_new:n \{g__tag_role/Namespace_#1_dict\}}
{\pdfdict_new:n \{g__tag_role/RoleMapNS_#1_dict\}}
{\pdfdict_gput:nnn \{g__tag_role/Namespace_#1_dict\}}
{\prop_gput:Nne \g__tag_role_NS_prop \{#1\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_prop \{#2\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_class_prop \{#3\}{}}
{\pdf_object_new:n \{tag/NS/#1\}}
{\pdfdict_new:n \{g__tag_role/Namespace_#1_dict\}}
{\pdfdict_gput:nnn \{g__tag_role/Namespace_#1_dict\}}
{\prop_gput:Nne \g__tag_role_NS_prop \{#1\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_prop \{#2\}{}}
{\prop_gput:Nne \g__tag_role_NS_#1_class_prop \{#3\}{}}
{\pdf_object_new:n \{tag/NS/#1\}}
\end{verbatim}
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:Nn \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
\c__tag_role_NS_new:nnn {pdf} {http://iso.org/pdf/asn}{}
\c__tag_role_NS_new:nnn {pdf2} {http://iso.org/pdf2/asn}{}
\c__tag_role_NS_new:nnn {mathml} {http://www.w3.org/1998/Math/MathML}{}
\c__tag_role_NS_new:nnn {latex} {https://www.latex-project.org/ns/dflt/2022}{}
\c__tag_role_NS_new:nnn {latex-book} {https://www.latex-project.org/ns/book/2022}{}
\exp_args:Nn \c__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.allocatennn
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 tagnname, ns, type
{
\lua_now:e { ltx.__tag.func.alloctag ('#1') }
\prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
\prop_gput:cn {g__tag_role_NS_#2_prop} {#1}{\{}\ {}
\prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{#3}
\prop_gput:cn {g__tag_role_NS_#2_class_prop} {#1}{--UNUSED--}
}
\cs_generate_variant:Nn \_tag_role_alloctag:nnn {nnV}
(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
{
\prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
\prop_gput:cn {g__tag_role_NS_#2_prop} {#1}{\{}\{}
\prop_gput:Nnn \g__tag_role_tags_class_prop {#1}{--UNUSED--}
\prop_gput:cn {g__tag_role_NS_#2_class_prop} {#1}{#3}
}
\cs_generate_variant:Nn \_tag_role_add_tag:nn {nnV}

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.
now the addition
\prop_get:NNN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
\quark_if_no_value:NTF \l__tag_tmpa_tl{\tl_to_str:n{#2}}
\prop_gput:Nne \g__tag_role_rolemap_prop {#1}{\tl_to_str:n{#2}}
\prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
\cs_generate_variant:Nn \__tag_role_add_tag:nn {VV,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.
\pdf_version_compare:NnT < {2.0}
\cs_new:Npn \__tag_role_get:nnNN #1 tag, #2 NS, #3 tlvar which hold the role tag
\prop_get:NNN \g__tag_role_rolemap_prop {#2}\l__tag_tmpa_tl
\quark_if_no_value:NTF \l__tag_tmpa_tl{\tl_to_str:n{#2}}
\prop_gput:NnV \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
\cs_generate_variant:Nn \__tag_role_get:nnNN {VVNN}
(End of definition for \__tag_role_get:nnNN.)
1.3.2 The pdf 2.0 version

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}{#4} \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:nnV {#1}{#2}\l__tag_tmpa_tl

Do not remap standard tags. TODO add warning?

\tl_if_in:nnF {-pdf-pdf2-mathml-}{-#2-} \prop_get:cnN { g__tag_role_NS_#4_prop } {#3}{#4} \quark_if_no_value:NT \l__tag_tmpa_tl \{ \tl_set:Nn \l__tag_tmpa_tl {--UNKNOWN--} \} \__tag_role_alloctag:nnV {#1}{#2}\l__tag_tmpa_tl

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}{#4} \quark_if_no_value:NT \l__tag_tmpa_tl \prop_get:cne { g__tag_role_NS_#2_prop } {#1} \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 }
For the parent-child test we must be able to get the role. We use the same number of arguments as for the \textless 2.0 command (and assume that we don’t need a name space)\textbackslash\_tag_role_get:nnNN

\pdf_version_compare:NnF \textless {2.0}
\cs_new:Npn \_tag_role_get:nnNN #1#2#3#4
\prop_if_exist:cTF {g\_tag_role\_NS\_#2\_prop}
\prop_get:cnNTF {g\_tag_role\_NS\_#2\_prop} {#1}\_tag_get_tmpc_tl
\tl_set:Ne #3 {\exp_last_unbraced:NV\use_i:nn \_tag_get_tmpc_tl}
\tl_set:Ne #4 {\exp_last_unbraced:NV\use_ii:nn \_tag_get_tmpc_tl}
\msg_warning:nnn { tag } {role-unknown-tag} { #1 }
\tl_set:Nn #3 {#1}
\tl_set:Nn #4 {#2}
\cs_generate_variant:Nn \_tag_role_get:nnNN {VVNN}

(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.

This command will process a line in the name space file. The first argument is the name of the namespace. 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 %
\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}
\tl_set:Nn \l__tag_tmpa_tl {#5}
\__tag_role_alloctag:nnV {#2}{#1}\l__tag_tmpa_tl
\tl_if_eq:nnF {#2}{#3}
\__tag_role_add_tag:nn {#2}{#3}
\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 %
\tl_if_empty:nF { #2 }
\bool_if:NTF \l__tag_role_update_bool
{
\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

163
\_tag\_role\_alloctag:nn\textbackslash \textbackslash V \lbrace \#2\rbrace \lbrace \#1\rbrace \_tag\_tmpa\_tl
\bool\_lazy\_and:nnT
\lbrace ! \tl\_if\_empty\_p:n \lbrace \#3\rbrace \rbrace \lbrace ! \str\_if\_eq\_p:nn \lbrace \#1\rbrace \lbrace \textbackslash pdf2\rbrace \rbrace
\lbrace ! \tl\_if\_empty\_p:n \lbrace \#3\rbrace \rbrace \lbrace ! \str\_if\_eq\_p:nn \lbrace \#1\rbrace \lbrace \#3\rbrace \lbrace \#4\rbrace \rbrace
\prop\_gput:cn\lbrace g\_\_tag\_role\_NS\_\#1\_prop\rbrace \lbrace \#2\rbrace \lbrace \#3\rbrace \lbrace \#4\rbrace
\rbrace

(End of definition for \_\_tag\_role\_read\_namespace\_line:nn)

\_\_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
\lbrace
\prop\_if\_exist:cF \lbrace g\_\_tag\_role\_NS\_\#1\_prop\rbrace
\lbrace \msg\_warning:nnn \lbrace tag\rbrace \lbrace namespace\_unknown\rbrace \lbrace \#1\rbrace \rbrace
\rbrace
\lbrace ! \ior\_open:Nn \g\_tmpa\_ior \lbrace \textbackslash tagpdf\_ns\_\#2\_def\rbrace
\msg\_info:nnn \lbrace tag\rbrace \lbrace read\_namespace\rbrace \lbrace \#2\rbrace
\ior\_map\_inline:Nn \g\_tmpa\_ior
\lbrace
\_\_tag\_role\_read\_namespace\_line:nn \lbrace \#1\rbrace \#\#1,\ldots,\#\_q\_stop
\rbrace
\ior\_close:N\g\_tmpa\_ior
\rbrace
\rbrace

(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
\lbrace
\rbrace

(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.

in pdf 1.7 the following namespaces should only store the settings for later use:

But is the class provides a \chapter command then we switch

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.

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.
These two properties map the rule strings to numbers and back. There are in tagpdf-data.dtx near the csv files for easier maintenance.

\__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!

\cs_new_protected:Npn \__tag_store_parent_child_rule:nnn #1 #2 #3 % num parent, num child, #3 string
{
    \intarray_gset:Nnn \g__tag_role_parent_child_intarray { #1#2 }{0\prop_item:Nn\c__tag_role_rules_prop{#3}}
}

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} {
    \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:NVN \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 }
}
\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
\prop_get:NnN \g__tag_role_index_prop {mathml} {\l__tag_tmpa_tl}
\prop_get:NnN \g__tag_role_index_prop {math} {\l__tag_tmpa_tl}
\prop_map_inline:Nn \g__tag_role_NS_mathml_prop
\prop_gput:NnV \g__tag_role_index_prop{#1} {\l__tag_tmpa_tl}
\prop_gput:NnV \g__tag_role_index_prop{math} {\l__tag_tmpa_tl}

1.6.2 Retrieving the parent-child rule

This command retrieves the rule (as a number) and stores it in the tl-var. It assumes that the tag in #1 is a standard tag after role mapping for which a rule exist and is not one of Part, Div, NonStruct as the real parent has already been identified. #3 can be used to pass along data about the original tags and is only used in messages.

TODO check temporary variables. Check if the tl-var should be fix.

```
\tl_new:N \l__tag_parent_child_check_tll
\cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnnN #1 #2 #3 #4
% #1 parent (string) #2 child (string) #3 text for messages (eg. about Div or Rolemapping)
% #4 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#4
    { \intarray_item:Nn \g__tag_role_parent_child_intarray
      \{\l__tag_tmpa_tl\l__tag_tmpb_tl\} }
  }
}

Get the rule from the intarray
\tl_set:Ne\l__tag_tmpa_tl
% #4 = \prop_item:Nn\c__tag_role_rules_prop{} %warn ?
we must take the current child from the stack if is already there, depending on location
the check is called, this could also remove the parent, but that is ok too.
}
```

This is the message, this can perhaps go into debug mode.

```
\group_begin:
\int_compare:nNnT {\l__tag_tmapa_int*\l__tag_loglevel_int} > { 0 } 
{ \prop_get:NWF\c__tag_role_rules_num_prop #4 \l__tag_tmapa_tl
  \tl_set:Nn \l__tag_tmpa_tl {unknown}
  \tl_set:Nn \l__tag_tmpb_tl {#1}
  \msg_note:nneee
    { tag }
    { role-parent-child }
    { #1 }
    { #2 }
    { #4~(='\l__tag_tmpa_tl') }
  \iow_newline:
\group_end:
```

168
\group_end:
\group_end:
\group_end:
\group_end:
\tl_set:Nn\#4 {0}
\msg_warning:nnee
\{ tag 
\{role-parent-child
\{ #1
\{ #2
\{ unknown! 
\}
\}
\}
\}
\cs_generate_variant:Nn\__tag_role_get_parent_child_rule:nnnN {VVVN,VVnN}
(this definition for \__tag_role_get_parent_child_rule:nnnN)
This commands translates rolemaps its arguments and then calls \__tag_role_get_parent_child_rule:nnnN. It does not try to resolve inheritance of Div etc but instead warns that the rule can not be detected in this case. 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_check_parent_child:nnnnN #1 #2 #3 #4 #5
%#1 parent tag,#2 NS, #3 child tag, #4 NS, #5 tl var
\for debugging messages we store the arguments.
\prop_put:Nnn \l__tag_role_debug_prop {parent} {#1}
\prop_put:Nnn \l__tag_role_debug_prop {child} {#3}
get the standard tags through rolemapping if needed at first the parent
\prop_get:NnTF \g__tag_role_index_prop {#1}\l__tag_tmpa_tl
\{ \tl_set:Nn \l__tag_tmpa_tl {#1} \}
\prop_get:NnTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
\{ \tl_set:Nn \l__tag_tmpb_tl {\q_no_value} \}
now the child
\prop_get:NnTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
\{ \tl_set:Nn \l__tag_tmpb_tl {#3} \}
\prop_get:NnTF \g__tag_role_index_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

```latex
\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:VVnN
    \l__tag_tmpa_tl \l__tag_tmpb_tl
    \{Rolemapped-from:--'#1'-->--'#3'}
    #5
  
  \}
  \{
    \tl_set:Nn #5 {0}
    \msg_warning:nneee
    \{ tag \}
    \{role-parent-child\}
    \{ #1 \}
    \{ #3 \}
    \{ unknown! \}
  \}

\cs_new_protected:Npn \__tag_check_parent_child:nnN #1#2#3
  \{
    \__tag_check_parent_child:nnnnN {#1}{}{#2}{}#3
  \}
```

and now the pdf 2.0 version The version with three arguments retrieves the default names space and then calls the full command. Not sure if this will ever be needed but we leave it for now.

```latex
\cs_new_protected:Npn \__tag_check_parent_child:nnN #1 #2 #3
  \{
    \prop_get:NnN\g__tag_role_tags_NS_prop {#1}\l__tag_role_tag_namespace_tmpa_tl
    \prop_get:NnN\g__tag_role_tags_NS_prop {#2}\l__tag_role_tag_namespace_tmpb_tl
    \str_if_eq:nnT{#2}{MC}{\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl}
    \bool_lazy_and:nnTF
      \{ ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpa_tl \}
      \{ ! \quark_if_no_value_p:N \l__tag_role_tag_namespace_tmpb_tl \}
      \{
        \__tag_check_parent_child:nVnVN
        \l__tag_role_tag_namespace_tmpa_tl
        \l__tag_role_tag_namespace_tmpb_tl
        \#3
      
      \}
      \{
        \tl_set:Nn #3 {0}
        \msg_warning:nneee
        \{ tag \}
        \{role-parent-child\}
        \{ #1 \}
        \{ #2 \}
        \{ unknown! \}
      
      \}
  \}
```

170
and now the real command.

\cs_new_protected:Npn \__tag_check_parent_child:nnnN #1 #2 #3 #4 #5 \%tag,NS,tag,NS, tl var
{
\prop_put:Nnn \l__tag_role_debug_prop {parent} {#1/#2}
\prop_put:Nnn \l__tag_role_debug_prop {child} {#3/#4}

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 {#3}
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:VVnN
\l__tag_tmpa_tl \l__tag_tmpb_tl
(Rolemapped-from-‘#1/#2’---→-‘#3\str_if_empty:nF{#4}{/#4}’)
#5
\tl_set:Nn #5 {0}
\msg_warning:nneee
{ tag }
{role-parent-child}
{ #1 }
{ #3 }
{ unknown! }
\cs_generate_variant:Nn\__tag_check_parent_child:nnN {VVN}
\cs_generate_variant:Nn\__tag_check_parent_child:nnnnN {VVVVN,nVnVN,VVnnN}
⟨/package⟩

(End of definition for __tag_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}{}
{\seq_get:NN\g__tag_struct_stack_seq\l__tag_tmpa_tl
\__tag_struct_get_parentrole:eNN
\l__tag_tmpa_tl}
\l__tag_get_parent_tmpa_tl
\l__tag_get_parent_tmpb_tl
\__tag_check_parent_child:VVnN
\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}
1.7 Remapping of tags

In some context it can be necessary to remap or replace the tags. That means instead of tag=H1 or tag=section one wants the effect of tag=Span. Or instead of tag=P one wants tag=Code.

The following command provides some general interface for this. The core idea is that before a tag is set it is fed through a function that can change it. We want to be able to chain such functions, so all of them manipulate the same variables.

\_\_tag\_role\_remap:
This function is used in the structure and the mc code before using a tag. By default it does nothing with the tl vars. Perhaps this should be a hook?

\_\_tag\_role\_remap_id:
This is copy in case we have to restore the main command.

1.8 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.

\keys_define:nn { __tag / tag-role }
\keys_define:nn { __tag / setup }
\keys_set_known:nnnN \keys_set_known:nnnN

tag-name-space=user,
role-name-space=, %so that we can test for it.

#1
\{__tag/tag-role\} \l_tmpa_tl
\tl_if_empty:NF \l_tmpa_tl
{ \exp_args:NNno \seq_set_split:Nnn \l_tmpa_seq { / } {\l_tmpa_tl/} \tl_set:Ne \l__tag_role_tag_tmpa_tl { \seq_item:Nn \l_tmpa_seq {1} } \tl_set:Ne \l__tag_role_role_tmpa_tl { \seq_item:Nn \l_tmpa_seq {2} } }
\tl_if_empty:NT \l__tag_role_role_namespace_tmpa_tl
{ \prop_get:NVNTF \g__tag_role_tags_NS_prop \l__tag_role_role_tmpa_tl \l__tag_role_role_namespace_tmpa_tl
{ \prop_if_in:NVF \g__tag_role_NS_prop \l__tag_role_role_namespace_tmpa_tl
{ \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user} }
{ \tl_set:Nn \l__tag_role_role_namespace_tmpa_tl {user} }
}
\pdf_version_compare:NnTF < {2.0}
{ \tODO add check for emptiness?
\{ \l__tag_role_add_tag:VV \l__tag_role_tag_tmpa_tl \l__tag_role_role_tmpa_tl
\}
{ \l__tag_role_add_tag:VVVV \l__tag_role_tag_tmpa_tl \l__tag_role_tag_namespace_tmpa_tl \l__tag_role_role_tmpa_tl \l__tag_role_role_namespace_tmpa_tl
}
}
\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} }
deprecated names
\mathml-tags .bool_gset:N = \g__tag_role_add_mathml_bool
\add-new-tag .meta:n = {role/new-tag=#1}

(End of definition for tag (rolemap-key) and others. These functions are documented on page 154.)
Part X
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.

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.

show-spaces,(deprecated)

1 Code for interword spaces

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

activate/spaces,(setup-key)
interwordspace,(deprecated)
show-spaces,(deprecated)

1 (@@=tag)
2 (+header)
3 \ProvidesExplPackage {tagpdf-space-code} {2024-02-29} {0.98x}
4 {part of tagpdf - code related to real space chars}
5 (/header)

175
\_tag\_fakespace: For \texttt{luatex} we need a command for the fake space as equivalent of the \texttt{pdftex} primitive.
\sys_if_engine_luatex:T
{
\cs_new_protected:Nn \_tag_fakespace:
{
 \group_begin:
 \lua_now:e{ltx.__tag.func.fakespace()}
 \skip_horizontal:n{\c_zero_skip}
 \group_end:
}
}
⟨/package⟩

(End of definition for \_tag_fakespace:)
Index

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

Symbols

| \#       | 1034, 1038 |
| \  \    | 10, 23, 27, 28, 44, 45, 46, 53, 56, 58, 64, 66, 77, 256, 257, 258, 396, 459, 467 |
| \u      | 421, 432 |

A

activate_{\text{setup-key}} 35, 255
activate-all_{\text{setup-key}} 6
activate-mc_{\text{setup-key}} 6
activate-tree_{\text{setup-key}} 6
activate-mc_{\text{setup-key}} 6
activate/socket_{\text{setup-key}} 255
activate/space_{\text{setup-key}} 175
activate/spaces_{\text{setup-key}} 6
activate/struct_{\text{setup-key}} 247
activate/struct-dest_{\text{setup-key}} 247
activate/tagunmarked_{\text{setup-key}} 268
activate/tree_{\text{setup-key}} 247
actualtext_{\text{mc-key}} 69, 253, 453
actualtext_{\text{struct-key}} 453
add-new-tag_{\text{setup-key}} 700
add-new-tag_{\text{deprecated}} 680
add-new-tag_{\text{setup-key}} 154
\AddToHook 13, 16, 57, 87, 273, 356, 378, 509, 511, 512, 516, 520, 527, 556, 601
AF_{\text{struct-key}} 100, 615
AFinline_{\text{struct-key}} 100, 615
AFinline-o_{\text{struct-key}} 100, 615
APref_{\text{struct-key}} 100, 615
alt_{\text{mc-key}} 69, 253, 453
alt_{\text{struct-key}} 99, 504
artifact_{\text{mc-key}} 69, 253, 453
artifact-bool internal commands:
  _artifact-bool 121
artifact-type internal commands:
  _artifact-type 121
attr-unknown 20, 70
attribute_{\text{struct-key}} 101, 1156
attribute-class_{\text{struct-key}} 101, 1152

B

bool commands:
  \bool_gset_eq:NN 621, 636, 648, 666
  \bool_gset_true:N 47, 49, 55, 110, 117, 369
  \bool_if:nTF 6, 346
  \bool_if_exist_p:N 44
  \bool_lazy_all:nTF 102
  \bool_lazy_and:nnTF 43, 136, 146, 275, 311, 366, 454, 510, 528, 558, 635
  \bool_lazy_and_p:NN 8
  \bool_set_false:N 42, 44, 16, 178, 188, 189, 190, 211, 212, 213, 239, 346, 390, 593, 615, 642
  \bool_set_true:N 15, 112, 114, 198, 199, 200, 201, 222, 223, 235, 258, 348, 389, 592
  \box 375
  box commands:
    \box_dp:N 176, 180
    \box_ht:N 166
    \box_new:N 100, 101
    \box_set_dp:N 174, 176
    \box_set_eq:NN 189
    \box_set_ht:N 173, 175
    \box_use_drop:N 178, 182
  \boxmaxdepth 80, 177
  \c 269, 270
  c@g internal commands:
    \c@g__tag_MCID_abs_int 11, 15, 24, 33, 46, 53, 64, 70, 80, 134, 159, 180, 239, 242, 269, 274, 303, 344, 351, 416
    \c@g__tag_parenttree_obj_int 151, 455
    \c@g__tag_struct_abs_int 6, 18, 38, 81, 89, 112, 113, 145, 147, 150, 152, 229, 354, 480, 515, 537, 549, 563, 579, 587, 600, 610, 629, 632, 637, 671, 673, 678, 690, 692, 697, 750, 761, 762, 763, 764, 765, 766, 769, 771, 777, 780, 797, 804, 822

C

178
legacy commands:
\legacy_if:nTF 72, 462, 465, 466
\lap 421
\log_(deprecated) 255

ltx. internal commands:
ltx.__tag.func.alloca_tag 271
ltx.__tag.func.fakespace 445
ltx.__tag.func.fill_parent_tree_-line 812
ltx.__tag.func.get_num_from 280
ltx.__tag.func.get_tag_from 299
ltx.__tag.func.mark_page_-elements 643
ltx.__tag.func.mark_shipout 795
ltx.__tag.func.markspaceoff 511
ltx.__tag.func.markspaceon 511
ltx.__tag.func.mc.insert_kids 580
ltx.__tag.func.mc.num_of_kids 329
ltx.__tag.func.output_num_from 280
ltx.__tag.func.output_parenntree 812
ltx.__tag.func.output_tag_from 299
ltx.__tag.func.pdf_object_ref 425
ltx.__tag.func.space_chars_-shipout 543
ltx.__tag.func.store_mc_data 314
ltx.__tag.func.store_mc_in_page 624
ltx.__tag.func.store_mc_kid 322
ltx.__tag.func.store_mc_label 319
ltx.__tag.func.store_mc_struct_-mcabs 612
ltx.__tag.func.update_mc_-attributes 632
ltx.__tag.tables.role_tag_-attribute 269
ltx.__tag.trace.log 183
ltx.__tag.trace.show_all_mc_data 240
ltx.__tag.trace.show_mc_data 225
ltx.__tag.trace.show_prop 200
ltx.__tag.trace.show_seq 191
ltx.__tag.trace.show_struct_data 246

lua commands:
\msg_error:nnn 159, 180, 430, 794 \msg_error:nnn 196, 207, 215, 226, 261, 417, 1166, 1206 \msg_error:nnnnn 535, 544 \msg_info:nnn 134, 173, 174, 249, 253, 327, 335 \msg_info:nnnnn 203, 222 \msg_line_context: 77, 363, 364, 396, 400, 404, 460, 468 \g_msg_module_name_prop 30, 34 \g_msg_module_type_prop 33 \msg_new:nnn 7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 51, 60, 71, 72, 73, 74, 75, 76, 78, 80, 81, 82, 83, 84, 85, 87, 254, 303, 364, 394, 398, 402, 454, 462 \msg_new:nnnnn 90 \msg_note:nnn 169 \msg_note:nnn 186, 203, 384, 391, 426, 434 \msg_note:nnnnn 209, 226, 370, 377, 411, 419 \msg_note:nnnnnn 477 \msg_redirect_name:nnn 531 \msg_show_item_unbraced:nn 246 \msg_show_item_unbraced:n 237 \msg_term:nnnnnn 231, 240 \msg_warning:nnn 24, 212, 261 \msg_warning:nnnnn 11, 13, 41, 44, 53, 166, 189, 234, 242, 244, 250, 265, 288, 323, 604, 631, 1013, 1032, 1076 \msg_warning:nnnnnn 397, 447, 519 \msg_warning:nnnnnnn 217, 407, 492, 539, 569, 646, 860

\newcommand 589, 590 \newcounter 6, 8, 151 \NewDocumentCommand 6, 23, 29, 34, 40, 46, 51, 56, 94, 286, 594 \newmarks 13
\namespace_r{rolemap-key} 154
\msg_error:nnn 159, 180, 430, 794 \msg_error:nnn 196, 207, 215, 226, 261, 417, 1166, 1206 \msg_error:nnnnn 535, 544 \msg_info:nnn 134, 173, 174, 249, 253, 327, 335 \msg_info:nnnnn 203, 222 \msg_line_context: 77, 363, 364, 396, 400, 404, 460, 468 \g_msg_module_name_prop 30, 34 \g_msg_module_type_prop 33 \msg_new:nnn 7, 8, 9, 12, 13, 14, 15, 16, 22, 24, 25, 32, 35, 36, 38, 40, 42, 51, 60, 71, 72, 73, 74, 75, 76, 78, 80, 81, 82, 83, 84, 85, 87, 254, 303, 364, 394, 398, 402, 454, 462 \msg_new:nnnnn 90 \msg_note:nnn 169 \msg_note:nnnnn 186, 203, 384, 391, 426, 434 \msg_note:nnnnn 209, 226, 370, 377, 411, 419 \msg_note:nnnnnn 477 \msg_redirect_name:nnn 531 \msg_show_item_unbraced:nn 246 \msg_show_item_unbraced:n 237 \msg_term:nnnnnnn 231, 240 \msg_warning:nnn 24, 212, 261 \msg_warning:nnnnnnn 11, 13, 41, 44, 53, 166, 189, 234, 242, 244, 250, 265, 288, 323, 604, 631, 1013, 1032, 1076 \msg_warning:nnnnnnnn 397, 447, 519 \msg_warning:nnnnnnnn 217, 407, 492, 539, 569, 646, 860

\namespace_r{rolemap-key} 154
\new-tag 20, 80 \newattribute_r{deprecated} 101, 1134
\newcommand 589, 590 \newcounter 6, 8, 151 \NewDocumentCommand 6, 23, 29, 34, 40, 46, 51, 56, 94, 286, 594 \newmarks 13
\seq_new:N 12, 14, 14, 15, 16, 17, 17,
18, 18, 24, 96, 97, 134, 167, 766, 1131
\seq_pop_left:NN ... 411, 413, 414
\seq_put_right:NN ... 282
\seq_remove_all:NN ... 285
\seq_set_eq:NN ... 203, 204
\seq_set_from_clist:NN ... 1157, 1193
\seq_set_from_clist:NN

83, 86, 192, 212, 399, 410
\seq_set_map:NnN ... 280
\seq_set_map:e:NNn ... 1158, 1194
\seq_set_split:NNn

49, 124, 487, 495, 702
\seq_show:N 57, 172, 186, 187, 220,
283, 284, 286, 335, 840, 892, 913, 923
\seq_use:Nn ... 45,
106, 107, 201, 203, 204, 291, 344, 1173
\l_tapa_seq 305, 325, 333, 702, 703, 704
\setbox ............................................ 363
shipout commands:
\g_shipout_readonly_int

...... 79, 157, 231, 361
show-kids .................. 20, 50
show-spaces, (deprecated) .... 175, 0
show-struct .................. 20, 50
\ShowTagging .............. 17, 96, 93
skip commands:
\skip_horizontal:n ............ 76
\c_zero_skip ................. 76
socket commands:
\socket_assign_plug:nn ........... 503, 507, 508, 524, 738, 739
\socket_new:nn ............ 436, 437, 484
\socket_new_plug:nnn ............ 439, 458, 485, 491, 493
\socket_use:n ........... 509, 511, 518, 522
\socket_use:nn ............ 527
stash, (mc-key) .............. 69, 121
stash, (struct-key) ............ 99, 504
str commands:
\str_case:nnTF ............. 59, 809
\str_const:N ............. 50
\str_if_empty:nTF ........... 641
\str_if_eq:nTF ........... 123, 357, 443, 557
\str_if_eq:p:nn ........... 312, 348, 350
\str_new:N ............... 94
\str_set_convert:NNnn ....... 125, 278,
299, 467, 480, 531, 543, 557, 573, 604
\str_use:N ............ 289, 312
\string ................... 20, 21, 22, 578
struct-faulty-nesting ........... 20, 32
struct-label-unknown ........... 20, 38
struct-missing-tag ............ 20, 35
struct-no-objnum ............ 20, 24
struct-orphan .............. 20, 25
struct-show-closing ........... 20, 40
struct-stack,(show-key) ........... 36, 216
struct-unknown ............ 20, 22
struct-used-twice ........... 20, 36
\SuspendTagging .............. 39
sys commands:
\c_sys_backend_str ........... 59
\c_sys_engine_str ........... 11, 13
\sys_if_engine_luatex:TF ....... 48,
49, 70, 83, 103, 105, 116, 283, 284
\sys_if_engine_pdfTeX:TF ........... 25
\sys_if_output_pdf:TF ........... 11, 27
sys-no-interwordspace ........... 20, 87

T
\tagsetorder,(deprecated) ........... 6, 271
\tag,(mc-key) .............. 69, 255, 453
\tag,(rolemap-key) ............ 154, 580
\tag,(struct-key) ............ 99, 504
tag commands:
\tag_check_child:nn ........... 154, 658, 660
\tag_check_child:nTF ........... 154, 558
\tag_get:n ............... 17,
70, 98, 99, 114, 115, 88, 91, 94, 93, 396
\tag_if_active: .......... 96, 100
\tag_if_active:nTF ............ 17, 95, 529
\tag_if_active:p: ........... 17, 95, 367
\tag_if_box_tagged:n ............ 17, 119
\tag_if_box_tagged:nTF ............ 17, 118
\tag_if_box_tagged:p:n ............ 17, 118
\tag_mc_artifact_group_begin:n ........... 68, 59, 59, 62
\tag_mc_artifact_group_end: ........... 68, 59, 60, 70
\tag_mc_begin:n ............ 10, 68, 25, 65,
113, 171, 171, 351, 351, 355, 361,
420, 431, 455, 487, 624, 652, 703, 726
\tag_mc_begin_pop:n ............ 68,
75, 79, 80, 101, 633, 663, 717, 740
\tag_mc_end: .............. 68,
31, 74, 92, 233, 233, 351, 352, 422,
428, 433, 434, 497, 630, 659, 715, 738
\tag_mc_end:push: ............ 68, 64, 79, 79, 82, 618, 645, 701, 724
\tag_mc_if_in: ............. 80, 228
\tag_mc_if_in:nTF ............ 68, 42, 66, 221
\tag_mc_if_in:p: ............ 68, 42, 66, 221
\tag_mc_reset_box:N ............ 69, 78, 78, 245, 245
\tag_mc_use:n ............ 68, 35, 35, 36, 37
\l_tag_para_attr_class_tl ........... 380, 382
\socket_use:n ............ 38, 39, 62, 66, 67
\tag_socket_use:n ............ 38, 39, 63, 66, 72
\tag_start: ............. 6, 182, 193, 206, 231
\g_tag_tagunmarked_bool 116, 268, 270
\l_tag_tmpb_box .................... 89, 167, 173, 174, 178, 189, 190
\l_tag_tmpb_clist .................... 89, 1156, 1157, 1190, 1191, 1193
\l_tag_tmpb_int .................... 88, 89, 91, 96, 99, 103, 112, 386, 398, 400, 470
\l_tag_tmpb_prop .................... 89, 172, 180, 193, 195
\l_tag_tmpb_seq ..... 49, 56, 57, 89, 280, 282, 284, 285, 286, 287, 292, 399, 402, 410, 411, 413, 414, 415, 487, 488, 489, 495, 496, 497, 1158, 1162, 1172, 1173, 1174, 1176, 1194, 1200, 1202, 1226
\l_tag_tmpb_str .................... 42, 43, 48, 94, 279, 284, 289, 300, 305, 312, 468, 473, 481, 486, 532, 539, 544, 551, 558, 565, 574, 581, 605, 612
\l_tag_tmpb_tl .................... 89, 168, 175, 176, 180, 182
\l_tag_tmpb_seq .................... 89, 1157, 1158, 1193, 1194
\_tag_tree_fill_parenttree: .................... 167, 168, 243
\_tag_tree_final_checks: 20, 20, 342 \g_tag_tree_id_pad_int 76, 80, 143 \_tag_tree_lua_fill_parenttree: 223, 223, 240 \_tag_tree_openaction_struct-tl 30, 36, 55 \_tag_tree_parenttree_rerun-msg: 167, 210, 245 \_tag_tree_update_openaction: 40, 73 \_tag_tree_write_classmap: 276, 276, 346 \_tag_tree_write_idtree: 84, 344 \_tag_tree_write_namespaces: 310, 310, 347 \_tag_tree_write_parenttree: 296, 236, 343 \_tag_tree_write_rolemap: 254, 253, 345 \_tag_tree_write_structelements: 143, 143, 348 \_tag_tree_write_structree: 124, 124, 349 \_tag_tree_write_rolemap: 252 \_tag_tree_write_rolemap: 680 tag/namespace/(rolemap-key) 29 tag/struct/0 internal commands: \_tag/struct/0 \_tag/tree/namespaces internal commands: \_tag/tree/namespaces \_tag/tree/parenttree internal commands: \_tag/tree/parenttree \_tag/tree/rolemap internal commands: \_tag/tree/rolemap \_tag/abs/page 6, 145 \tag/cabs 6, 145 \tagsbegin 35, 155, 22, 369, 375 \tag/cend 35, 22, 375 \tag/cid 6, 145 \tag/cifin 35 \tag/cifin/T 35, 39 \tag/cuse 35, 22 \tag/pdf/para/bf 37, 589 \tag/pdf/para0n 37, 589 \tag/pdf/setup 35, 101, 154, 6 \tag/pdf/suppressmarks 37, 594 \tag/start 6, 206, 233 \tag/stop 6, 205, 232 \tag/struct \_tag/struct 6, 145 \tag/struct/begin \_tag/struct/end \_tag/structobj 36, 154, 155, 45, 258, 360, 362 \tag/struct/end 36, 45, 259, 375 \tag/struct/obj 6, 145 \tag/structure 36, 45 \tag/tool 35, 15 \tag/unmarked/(deprecated) 6, 208 \text{TeX} and \text{LaTeX} \text{2e} commands: \text{"\LaTeX"} 164 \text{"\LaTeX\TeX\LaTeX"} 74
<table>
<thead>
<tr>
<th>U</th>
<th>\UseTaggingSocket</th>
<th>\vbadness</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>uncompress</td>
<td>255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unittag</td>
<td>386</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\unskip</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>use commands:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\use:N</td>
<td>94, 595</td>
<td>\vbox_set_split_to_ht:NNn</td>
<td>190</td>
</tr>
<tr>
<td>\use:n</td>
<td>41, 309</td>
<td>\vbox_set_to_ht:Nnn</td>
<td>166</td>
</tr>
<tr>
<td>\use_i:nn</td>
<td></td>
<td>\vbox_unpack_drop:N</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>184, 240, 361, 375, 445, 449, 944</td>
<td>\vfuzz</td>
<td>165</td>
</tr>
<tr>
<td>\use_ii:nn</td>
<td>185, 241, 332</td>
<td>\viewer/startpage,(setup-key)</td>
<td>32</td>
</tr>
<tr>
<td>\use_none:n</td>
<td>77, 86, 89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\use_none:nn</td>
<td>76, 87, 1091</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\UseSocket</td>
<td>38, 39, 70, 75, 80</td>
<td>\wd</td>
<td>373</td>
</tr>
</tbody>
</table>