tagpdf – A package to experiment with pdf tagging

Ulrike Fischer†
 Released 2022-12-22

Contents

1 Initialization and test if pdfmanagement is active. 7
2 Package options 7
3 Packages 8
  3.1 a LastPage label . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
4 Variables 9
5 Variants of l3 commands 10
6 Setup label attributes 11
7 Label commands 11
8 Commands to fill seq and prop 12
9 General tagging commands 12
10 Keys for tagpdfsetup 13
11 loading of engine/more dependent code 14

I The tagpdf-checks module
Messages and check code
Part of the tagpdf package 16
1 Commands 16

*This file describes v0.98, last revised 2022-12-22.
†E-mail: fischer@troubleshooting-tex.de
2 Description of log messages 16
   2.1 \ShowTagging command .............................................. 16
   2.2 Messages in checks and commands .................................. 16
   2.3 Messages from the ptagging code .................................. 17
   2.4 Warning messages from the lua-code .............................. 17
   2.5 Info messages from the lua-code .................................. 17
   2.6 Debug mode messages and code .................................... 18
   2.7 Messages ............................................................. 18

3 Messages 19
   3.1 Messages related to mc-chunks .................................... 19
   3.2 Messages related to structures .................................... 20
   3.3 Attributes ............................................................ 21
   3.4 Roles ................................................................. 21
   3.5 Miscellaneous ....................................................... 22

4 Retrieving data 22

5 User conditionals 23

6 Internal checks 23
   6.1 checks for active tagging ........................................ 23
   6.2 Checks related to structures ..................................... 24
   6.3 Checks related to roles .......................................... 25
   6.4 Check related to mc-chunks ...................................... 26
   6.5 Checks related to the state of MC on a page or in a split stream 28

II The tagpdf-user module 32
Code related to \LaTeX2e user commands and document commands
Part of the tagpdf package 32

1 Setup commands 32

2 Commands related to mc-chunks 32

3 Commands related to structures 33

4 Debugging 33

5 Extension commands 33
   5.1 Fake space ......................................................... 34
   5.2 Paratagging ......................................................... 34
   5.3 Header and footer ............................................... 34
   5.4 Link tagging ....................................................... 35

6 User commands and extensions of document commands 35

7 Setup and preamble commands 35

8 Commands for the mc-chunks 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9  Commands for the structure</td>
<td>36</td>
</tr>
<tr>
<td>10 Debugging</td>
<td>36</td>
</tr>
<tr>
<td>11 Commands to extend document commands</td>
<td>39</td>
</tr>
<tr>
<td>11.1 Document structure</td>
<td>39</td>
</tr>
<tr>
<td>11.2 Structure destinations</td>
<td>40</td>
</tr>
<tr>
<td>11.3 Fake space</td>
<td>40</td>
</tr>
<tr>
<td>11.4 Paratagging</td>
<td>41</td>
</tr>
<tr>
<td>11.5 Header and footer</td>
<td>43</td>
</tr>
<tr>
<td>11.6 Links</td>
<td>45</td>
</tr>
<tr>
<td>III The <strong>tagpdf-tree</strong> module</td>
<td>47</td>
</tr>
<tr>
<td>Commands trees and main dictionaries</td>
<td></td>
</tr>
<tr>
<td>Part of the tagpdf package</td>
<td></td>
</tr>
<tr>
<td>1 Trees, pdfmanagement and finalization code</td>
<td>47</td>
</tr>
<tr>
<td>1.1 Check structure</td>
<td>47</td>
</tr>
<tr>
<td>1.2 Catalog: MarkInfo and StructTreeRoot</td>
<td>48</td>
</tr>
<tr>
<td>1.3 Writing structure elements</td>
<td>48</td>
</tr>
<tr>
<td>1.4 ParentTree</td>
<td>49</td>
</tr>
<tr>
<td>1.5 Rolemap dictionary</td>
<td>51</td>
</tr>
<tr>
<td>1.6 Classmap dictionary</td>
<td>52</td>
</tr>
<tr>
<td>1.7 Namespaces</td>
<td>53</td>
</tr>
<tr>
<td>1.8 Finishing the structure</td>
<td>54</td>
</tr>
<tr>
<td>1.9 StructParents entry for Page</td>
<td>54</td>
</tr>
<tr>
<td>IV The <strong>tagpdf-mc-shared</strong> module</td>
<td>55</td>
</tr>
<tr>
<td>Code related to Marked Content (mc-chunks), code shared by all modes</td>
<td></td>
</tr>
<tr>
<td>Part of the tagpdf package</td>
<td></td>
</tr>
<tr>
<td>1 Public Commands</td>
<td>55</td>
</tr>
<tr>
<td>2 Public keys</td>
<td>56</td>
</tr>
<tr>
<td>3 Marked content code – shared</td>
<td>56</td>
</tr>
<tr>
<td>3.1 Variables and counters</td>
<td>57</td>
</tr>
<tr>
<td>3.2 Functions</td>
<td>58</td>
</tr>
<tr>
<td>3.3 Keys</td>
<td>60</td>
</tr>
<tr>
<td>V  The <strong>tagpdf-mc-generic</strong> module</td>
<td>62</td>
</tr>
<tr>
<td>Code related to Marked Content (mc-chunks), generic mode</td>
<td></td>
</tr>
<tr>
<td>Part of the tagpdf package</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1. Marked content code – generic mode</td>
<td>62</td>
</tr>
<tr>
<td>1.1 Variables</td>
<td>62</td>
</tr>
<tr>
<td>1.2 Functions</td>
<td>63</td>
</tr>
<tr>
<td>1.3 Looking at MC marks in boxes</td>
<td>66</td>
</tr>
<tr>
<td>1.4 Keys</td>
<td>73</td>
</tr>
<tr>
<td>VI The tagpdf-mc-luacode module</td>
<td>75</td>
</tr>
<tr>
<td>Code related to Marked Content (mc-chunks), luamode-specific</td>
<td></td>
</tr>
<tr>
<td>Part of the tagpdf package</td>
<td></td>
</tr>
<tr>
<td>1 Marked content code – luamode code</td>
<td>75</td>
</tr>
<tr>
<td>1.1 Commands</td>
<td>76</td>
</tr>
<tr>
<td>1.2 Key definitions</td>
<td>80</td>
</tr>
<tr>
<td>VII The tagpdf-struct module</td>
<td>83</td>
</tr>
<tr>
<td>Commands to create the structure</td>
<td></td>
</tr>
<tr>
<td>Part of the tagpdf package</td>
<td></td>
</tr>
<tr>
<td>1 Public Commands</td>
<td>83</td>
</tr>
<tr>
<td>2 Public keys</td>
<td>84</td>
</tr>
<tr>
<td>2.1 Keys for the structure commands</td>
<td>84</td>
</tr>
<tr>
<td>2.2 Setup keys</td>
<td>86</td>
</tr>
<tr>
<td>3 Variables</td>
<td>86</td>
</tr>
<tr>
<td>3.1 Variables used by the keys</td>
<td>88</td>
</tr>
<tr>
<td>3.2 Variables used by tagging code of basic elements</td>
<td>89</td>
</tr>
<tr>
<td>4 Commands</td>
<td>89</td>
</tr>
<tr>
<td>4.1 Initialization of the StructTreeRoot</td>
<td>90</td>
</tr>
<tr>
<td>4.2 Filling in the tag info</td>
<td>90</td>
</tr>
<tr>
<td>4.3 Handlings kids</td>
<td>91</td>
</tr>
<tr>
<td>4.4 Output of the object</td>
<td>94</td>
</tr>
<tr>
<td>5 Keys</td>
<td>97</td>
</tr>
<tr>
<td>6 User commands</td>
<td>102</td>
</tr>
<tr>
<td>7 Attributes and attribute classes</td>
<td>107</td>
</tr>
<tr>
<td>7.1 Variables</td>
<td>107</td>
</tr>
<tr>
<td>7.2 Commands and keys</td>
<td>107</td>
</tr>
<tr>
<td>VIII The tagpdf-luatex.def Driver for luatex</td>
<td>110</td>
</tr>
<tr>
<td>Part of the tagpdf package</td>
<td></td>
</tr>
<tr>
<td>1 Loading the lua</td>
<td>110</td>
</tr>
<tr>
<td>2 Logging functions</td>
<td>114</td>
</tr>
</tbody>
</table>
3 Helper functions
  3.1 Retrieve data functions ........................................ 116
  3.2 Functions to insert the pdf literals ........................... 118

4 Function for the real space chars .............................. 119

5 Function for the tagging ........................................ 122

6 Parenttree ......................................................... 127

IX The tagpdf-roles module
Tags, roles and namespase code
Part of the tagpdf package ........................................ 129

1 Code related to roles and structure names ................. 129
  1.1 Variables ..................................................... 129
  1.2 Namespaces .................................................. 132
  1.3 Adding a new tag ............................................ 133
    1.3.1 pdf 1.7 and earlier .................................... 134
    1.3.2 The pdf 2.0 version .................................... 135
  1.4 Helper command to read the data from files ............ 136
  1.5 Reading the default data .................................. 138
  1.6 Parent-child rules ........................................... 139
    1.6.1 Reading in the csv-files .............................. 139
    1.6.2 Retrieving the parent-child rule ..................... 141
  1.7 Remapping of tags .......................................... 146
  1.8 Key-val user interface ..................................... 147

X The tagpdf-space module
Code related to real space chars
Part of the tagpdf package ....................................... 149

1 Code for interword spaces ..................................... 149

Index ............................................................... 152
This is a temporary definition which will have to move to \l3ref. It allows to locally set a default value if the label or the attribute doesn’t exist. See issue \#4 in Accessible-xref.

We need commands to stop tagging in some places. There simply switches the two local booleans. The grouping commands can be used to group the effect.

This commands are intended as a pair. The start command will only restart tagging if the previous stop command with the same label actually stopped tagging.

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

activate-mc, activate-tree, activate-struct, activate-all are attributes used by the label/ref system.

activate-space\((setup-key)\)
activate-mc\((setup-key)\)
activate-tree\((setup-key)\)
activate-struct\((setup-key)\)
activate-all\((setup-key)\)

Keys to activate the various tagging steps

no-struct-dest\((setup-key)\)

The key allows to suppress the creation of structure destinations.

log\((setup-key)\)

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

tagunmarked\((setup-key)\)

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

tabsorder\((setup-key)\)

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

tagstruct
tagstructobj
tagabspage
tagmcabs
tagmcid
1 Initialization and test if pdfmanagement is active.

\begin{verbatim}
1 (\@=tag)
2 (%package)
3 \ProvidesExplPackage {tagpdf} {2022-12-22} {0.98}
4 { A package to experiment with pdf tagging }
5 \bool_if:nF
6 { \bool_lazy_and_p:nn
7   {\cs_if_exist_p:N \pdfmanagement_if_active_p:}
8   { \pdfmanagement_if_active_p: }
9 } 
10 { %error for now, perhaps warning later. 
11 \PackageError{tagpdf}
12 { PDF-resource-management-is-no-active!\MessageBreak
tagpdf-will-no-work. }
13 } 
14 { Activate-it-with \MessageBreak \string\RequirePackage{pdfmanagement-testphase}\MessageBreak \string\DocumentMetadata{<options>}\MessageBreak 
15 before-\string\documentclass
16 }
17 }
18 \end{verbatim} %/package

<*>debug>

\begin{verbatim}
26 \ProvidesExplPackage {tagpdf-debug} {2022-12-22} {0.98}
27 { debug code for tagpdf }
28 \IfPackageLoaded{tagpdf}{\PackageWarning{tagpdf-debug}{tagpdf-not-loaded,-quitting}\endinput}
</debug>

\end{verbatim}

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

\begin{verbatim}
36 (\@=tagpdf)
37 (%package)
38 \prop_gput:Nnn \g_msg_module_name_prop { tag }{ tagpdf }
39 \end{verbatim} %/package

Debug mode has its special mapping:

\begin{verbatim}
43 (%debug)
44 \prop_gput:Nnn \g_msg_module_type_prop { tag / debug} {}
45 \prop_gput:Nnn \g_msg_module_name_prop { tag / debug }{tagpdf-DEBUG}
46 </debug>

\end{verbatim}

2 Package options

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

\begin{verbatim}
49 (%package)
50 \bool_new:N\g__tag_mode_lua_bool
54 \DeclareOption {luamode} { \sys_if_engine_luatex:T { \bool_gset_true:N \g__tag_mode_lua_bool }
55 \DeclareOption {genericmode}{ \bool_gset_false:N\g__tag_mode_lua_bool }
56 \ExecuteOptions{luamode}
57 \ProcessOptions
\end{verbatim}
3 Packages

We need the temporary version of l3ref until this is in the kernel.
\RequirePackage{l3ref-tmp}

To be on the safe side for now, load also the base definitions
\RequirePackage{tagpdf-base}
⟨/package⟩
⟨∗base⟩
\ProvidesExplPackage{tagpdf-base}{2022-12-22}{0.98}
⟨part of tagpdf - provide base, no-op versions of the user commands ⟩
⟨/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:
⟨∗base⟩
\AddToHook{begindocument}{
\str_case:VnF \c_sys_backend_str { lualatex } { \cs_new_protected:Npn \__tag_whatsits: {} } \dvisvgm } { \cs_new_protected:Npn \__tag_whatsits: {} }
\{ \cs_new_protected:Npn \__tag_whatsits: {\tex_special:D {} } \}
⟨/base⟩

3.1 a LastPage label

See also issue #2 in Accessible-xref
\__tag_lastpagelabel:
⟨∗package⟩
\cs_new_protected:Npn \__tag_lastpagelabel:
{ \legacy_if:nT { @filesW }
 { \exp_args:NNnx \exp_args:NNx\iow_now:Nn \@auxout
 { \token_to_str:N \newlabeldata
 {\__tag_LastPage}
 {\__tag_LastPage}
 {\int_use:N \g_shipout_readonly_int}
 {\int_use:N \c@\__tag_MCID_abs_int }
 {\__tag_lastpagelabel:}
This allows to locally set a default value if the label or the attribute doesn’t exist.

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

(End definition for \ref_value:nnn. This function is documented on page 6.)

## 4 Variables

- \l__tag_tmpa_tl
- \l__tag_tmpb_tl
- \l__tag_get_tmpc_tl
- \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

(End definition for \l__tag_tmpa_tl and others."

- \c__tag_refmc_clist
- \c__tag_refstruct_clist

(End definition for \c__tag_refmc_clist and \c__tag_refstruct_clist.)

- \l__tag_loglevel_int

This integer hold the log-level and so allows to control the messages. TODO: a list which
log-level shows what is needed. The current behaviour is quite ad-hoc.

(End definition for \l__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{\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 if pdf version 2.0 is detected, but with the boolean struct-dest-boolean one can suppress them. Also we assume currently that they are set only at begin document. But if some control passing over groups are needed they could be perhaps used in a document too. TODO: check if they are used everywhere as needed and as wanted.

\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_gset_true:N \g__tag_active_struct_dest_bool
\end{verbatim}

These booleans should help to control the \textit{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
\end{verbatim}

This boolean controls if the code should try to automatically tag parts not in mc-chunk. It is currently only used in lua mode. It would be possible to use 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 \l3 commands}

\begin{verbatim}
\prg_generate_conditional_variant:Nnn \pdf_object_if_exist:n {e}{T,F}
\cs_generate_variant:Nn \pdf_object_ref:n {e}
\cs_generate_variant:Nn \pdffontchar:nnn {m,n}
\cs_generate_variant:Nn \pdffontdict:nnn {m,n}
\cs_generate_variant:Nn \pdffontdesc:nnn {m,n}
\cs_generate_variant:Nn \pdffontinfo:nnn {m,n}
\cs_generate_variant:Nn \pdffontname:nnn {m,n}
\cs_generate_variant:Nn \pdffontstretch:nnn {m,n}
\cs_generate_variant:Nn \pdffonttimescale:nnn {m,n}
\cs_generate_variant:Nn \pdffontweight:nnn {m,n}
\cs_generate_variant:Nn \pdffontchar:nnn {m,n}
\cs_generate_variant:Nn \pdffontdict:nnn {m,n}
\cs_generate_variant:Nn \pdffontdesc:nnn {m,n}
\cs_generate_variant:Nn \pdffontinfo:nnn {m,n}
\cs_generate_variant:Nn \pdffontname:nnn {m,n}
\cs_generate_variant:Nn \pdffontstretch:nnn {m,n}
\cs_generate_variant:Nn \pdffonttimescale:nnn {m,n}
\cs_generate_variant:Nn \pdffontweight:nnn {m,n}
\cs_generate_variant:Nn \pdffontchar:nnn {m,n}
\cs_generate_variant:Nn \pdffontdict:nnn {m,n}
\cs_generate_variant:Nn \pdffontdesc:nnn {m,n}
\cs_generate_variant:Nn \pdffontinfo:nnn {m,n}
\cs_generate_variant:Nn \pdffontname:nnn {m,n}
\cs_generate_variant:Nn \pdffontstretch:nnn {m,n}
\cs_generate_variant:Nn \pdffonttimescale:nnn {m,n}
\cs_generate_variant:Nn \pdffontweight:nnn {m,n}
\end{verbatim}
6 Setup label attributes

This are attributes used by the label/ref system. With structures we store the structure number `tagstruct` and the object reference `tagstructobj`. The second is needed to be able to reference a structure which hasn’t been created yet. The alternative would be to create the object in such cases, but then we would have to check the object existence all the time.

With mc-chunks we store the absolute page number `tagabspage`, the absolute id `tagmcabc`, and the id on the page `tagmcid`.

```latex
\ref_attribute_gset:nnnn { tagstruct } {0} { now } \int_use:N \c@g__tag_struct_abs_int
\ref_attribute_gset:nnnn { tagstructobj } {} { now } \int_use:N \c@g__tag_struct_abs_int
\ref_attribute_gset:nnnn { tagabspage } {0} { shipout } \int_use:N \g_shipout_readonly_int
\ref_attribute_gset:nnnn { tagmcabs } {0} { now } \int_use:N \c@g__tag_MCID_abs_int
\ref_attribute_gset:nnnn {tagmcid } {0} { now } \int_use:N \g__tag_MCID_tmp_bypage_int
```

(End definition for `tagstruct` and others. These functions are documented on page 6.)

7 Label commands

\__tag_ref_label:nn
A version of `\ref_label:nn` to set a label which takes a keyword `mc` or `struct` to call the relevant lists. TODO: check if `\@bsphack` and `\@esphack` make sense here.

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

(End definition for `\__tag_ref_label:nn`)

\__tag_ref_value:nnn
A local version to retrieve the value. It is a direct wrapper, but to keep naming consistent.... It uses the variant defined temporarily above.

```latex
\cs_new:Npn \__tag_ref_value:nnn #1 #2 #3 %#1 label, #2 attribute, #3 default
{ \ref_value:nnn {#1}{#2}{#3} }
\cs_generate_variant:Nn \__tag_ref_value:nnn {enn}
```

(End definition for `\__tag_ref_value:nnn`)

11
A command to retrieve the lastpage label, this will be adapted when there is a proper, kernel lastpage label.

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

(End definition for \__tag_ref_value_lastpage:nn.)

8 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.

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

\cs_generate_variant:Nn \__tag_prop_gput:Nnn { Nxn, Nxx, Nnx, cnn, cxxn, cno}
\cs_generate_variant:Nn \__tag_seq_gput_right:Nn { Nx, No, cn, cx}
\cs_generate_variant:Nn \__tag_prop_new:N { c }
\cs_generate_variant:Nn \__tag_seq_new:N { c }
\cs_generate_variant:Nn \__tag_seq_show:N { c }
\cs_generate_variant:Nn \__tag_prop_show:N { c }

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

9 General tagging commands

We need commands to stop tagging in some places. This simply switches the two local booleans. In some cases tagging should only restart, if it actually was stopped before. For this it is possible to label a stop.

\cs_new_protected:Npn \tag_stop_group_begin:
\group_begin:
\bool_set_false:N \l__tag_active_struct_bool
\bool_set_false:N \l__tag_active_mc_bool
\group_end:
\cs_set_eq:NN \tag_stop_group_end: \group_end:
\cs_set_protected:Npn \tag_stop:
\bool_set_false:N \l__tag_active_struct_bool
\bool_set_false:N \l__tag_active_mc_bool
\cs_set_protected:Npn \tag_start:
\cs_set_protected:Npn \tag_start:n
\cs_new_protected:Npn \tag_stop_group_begin:
\group_begin:
\bool_set_false:N \l__tag_active_struct_bool
\bool_set_false:N \l__tag_active_mc_bool
\group_end:
\cs_set_protected:Npn \tag_stop:
\bool_set_false:N \l__tag_active_struct_bool
\bool_set_false:N \l__tag_active_mc_bool
\cs_set_protected:Npn \tag_start:
\cs_set_protected:Npn \tag_start:n

(End definition for \__tag_prop_new:N and others.)
\bool_set_true:N \l__tag_active_struct_bool
\bool_set_true:N \l__tag_active_mc_bool
\prop_new:N\g__tag_state_prop
\cs_set_protected:Npn \tag_stop:n #1
{\tag_if_active:TF
 { \bool_set_false:N \l__tag_active_struct_bool
 \bool_set_false:N \l__tag_active_mc_bool
 \prop_gput:Nnn \g__tag_state_prop { #1 }{ 1 } }
 { \prop_gremove:Nn \g__tag_state_prop { #1 } }
}
\cs_set_protected:Npn \tag_start:n #1
{ \prop_gpop:NnN \g__tag_state_prop {#1}\l__tag_tmpa_tl
 \quark_if_no_value:NF \l__tag_tmpa_tl
 { \bool_set_true:N \l__tag_active_struct_bool
 \bool_set_true:N \l__tag_active_mc_bool
 }
}
\cs_new_protected:Npn \tag_stop:{}
\cs_new_protected:Npn \tag_start:{}
\cs_new_protected:Npn \tag_stop:n{}
\cs_new_protected:Npn \tag_start:n{}
\keys_define:nn { __tag / setup }
{ activate-space .bool_gset:N = \g__tag_active_space_bool,
 activate-mc .bool_gset:N = \g__tag_active_mc_bool,
 activate-tree .bool_gset:N = \g__tag_active_tree_bool,
 activate-struct .bool_gset:N = \g__tag_active_struct_bool,
 activate-all .meta:n =
 {activate-mc=#1,activate-tree=#1,activate-struct=#1},

(End definition for \tag_stop_group_begin: and others. These functions are documented on page 6.)

10 Keys for tagpdfsetup

TODO: the log-levels must be sorted

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

(*package)
\keys_define:nn { __tag / setup }
{ activate-space .bool_gset:N = \g__tag_active_space_bool,
 activate-mc .bool_gset:N = \g__tag_active_mc_bool,
 activate-tree .bool_gset:N = \g__tag_active_tree_bool,
 activate-struct .bool_gset:N = \g__tag_active_struct_bool,
 activate-all .meta:n =
 {activate-mc=#1,activate-tree=#1,activate-struct=#1},

(End definition for \tag_stop_group_begin: and others. These functions are documented on page 6.)
activate-all .default:n = true,
no-struct-dest .bool_gset_inverse:N = \g__tag_active_struct_dest_bool,

(End definition for activate-space (setup-key) and others. These functions are documented on page 6.)

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

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

(End definition for log (setup-key). This function is documented on page 6.)

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

(End definition for tagunmarked (setup-key). This function is documented on page 6.)
tabsorder,(setup-key) This sets the tabsorder on a page. The values are row, column, structure (default) or none. Currently this is set more or less globally. More finer control can be added if needed.
tabsorder .choice:,
tabsorder / row .code:n = \pdfmanagement_add:nnn { Page } {Tabs}{/R},
tabsorder / column .code:n = \pdfmanagement_add:nnn { Page } {Tabs}{/C},
tabsorder / structure .code:n = \pdfmanagement_add:nnn { Page } {Tabs}{/S},
tabsorder / none .code:n = \pdfmanagement_remove:nn {Page} {Tabs},
tabsorder .initial:n = structure,
uncompress .code:n = { \pdf_uncompress: },

(End definition for tabsorder (setup-key). This function is documented on page 6.)

11 loading of engine/more dependent code
\sys_if_engine_luatex:T
{ \file_input:n {tagpdf-luatex.def} }
{/package}
\bool_if:NTF \g__tag_mode_lua_bool
{
\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} \%
}
}
Part I

The tagpdf-checks module

Messages and check code

Part of the tagpdf package

1 Commands

```latex
\tag_if_active_p: * \tag_if_active:IF *
```

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.

```latex
\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 and struct_tag and struct_num.

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>\ShowTagging mc-data = num</td>
<td>log+term</td>
<td>lua-only</td>
</tr>
<tr>
<td>\ShowTagging mc-current</td>
<td>log+term</td>
<td></td>
</tr>
<tr>
<td>\ShowTagging struck-stack= [log</td>
<td>show]</td>
<td>log or term+stop</td>
</tr>
</tbody>
</table>

2.2 Messages in checks and commands

<table>
<thead>
<tr>
<th>command</th>
<th>message</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0_check_structure_has_tag:n</td>
<td>struct-missing-tag</td>
<td>error</td>
</tr>
<tr>
<td>#0_check_structure_tag:N</td>
<td>role-unknown-tag</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_info_closing_struct:n</td>
<td>struct-show-closing</td>
<td>info</td>
</tr>
<tr>
<td>#0_check_no_open_struct:</td>
<td>struct-faulty-nesting</td>
<td>error</td>
</tr>
<tr>
<td>#0_check_add_tag_role:nn</td>
<td>role-missing, role-tag, role-unknown</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_sc_if_nested:</td>
<td>mc-nested</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_sc_if_open:</td>
<td>mc-not-open</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_sc_pushed_popped:nn</td>
<td>mc-pushed, mc-popped</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_sc_tag:N</td>
<td>mc-tag-missing, role-unknown-tag</td>
<td>error (missing), warning (unknown).</td>
</tr>
<tr>
<td>#0_check_sc_used:n</td>
<td>mc-used-twice</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_show_MCID_by_page:</td>
<td>mc-label-unknown, mc-used-twice</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_struct_used:n</td>
<td>sys-no-intervspace</td>
<td>info (&gt;0)</td>
</tr>
<tr>
<td>#0_check_struct_use:n</td>
<td>struct-no-objnum</td>
<td>warning</td>
</tr>
<tr>
<td>#0_check_fat:</td>
<td>struct-faulty-nesting</td>
<td>error</td>
</tr>
<tr>
<td>#0_check_fat_used:n</td>
<td>struct-label-unknown</td>
<td>warning</td>
</tr>
<tr>
<td>#0_struct_write_obj:n</td>
<td>attr-unknown</td>
<td>error</td>
</tr>
<tr>
<td>#0_struct_begin:n</td>
<td>tree-acid-index-wrong</td>
<td>error</td>
</tr>
<tr>
<td>#0_struct_insert_annot:n</td>
<td>para-book-count-wrong</td>
<td>warning</td>
</tr>
</tbody>
</table>

In enddocument/info-hook

16
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></td>
</tr>
<tr>
<td>WARN TAG-OPEN-MC:</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WARN SHIPOUT-MC-OPEN:</td>
<td>1</td>
<td></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-ENDHEAD-REV</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-HEAD</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-INSERT-ARTIFACT</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-INSERT-BDC</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-INSERT-EMC</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-INSERT-TAG</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-KERN-SUBTYPE</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-MATH-SUBTYPE</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-MC-COMpare</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-MC-INTO-PAGE</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-NEW-MC-NODE</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-NODE</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-NO-HEAD</td>
<td>3</td>
<td>replaced by artifact</td>
</tr>
<tr>
<td>INFO TAG-NOT-TAGGED</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-QUITTING-BOX</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-STORE-MC-KID</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-TRVERSING-BOX</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-USE-ACTUALTEXT</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-USE-ALT</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TAG-USE-RAW</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>INFO TEX-MC-INSERT-KID</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
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</th>
<th>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>
<td></td>
</tr>
<tr>
<td></td>
<td>mc-begin-ignore</td>
<td>msg</td>
<td>if inactive</td>
</tr>
</tbody>
</table>

2.7 Messages

---

**mc-nested**

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

**mc-tag-missing**

**mc-label-unknown**

**mc-used-twice**

**mc-not-open**

**mc-pushed**

**mc-popped**

**mc-current**

---

**struct-no-objnum**

Various messages related to structure. TODO document their meaning.

**struct-faulty-nesting**

**struct-missing-tag**

**struct-used-twice**

**struct-label-unknown**

**struct-show-closing**

---

**attr-unknown**

Message if an attribute is unknown.

---

**role-missing**

**role-unknown**

**role-unknown-tag**

**role-tag**

**new-tag**

---

Message related to role mapping.
### 3 Messages

#### 3.1 Messages related to mc-chunks

**mc-nested** This message is issue is a mc is opened before the previous has been closed. This is not relevant for luamode, as the attributes don’t care about this. It is used in the `\check_mc_if_nested: test.`

```
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
(part of tagpdf - code related to checks, conditionals, debugging and messages)
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
\ProvidesExplPackage {tagpdf-checks-code} {2022-12-22} {0.98}
```

(End definition for mc-nested. This function is documented on page 18.)

**mc-tag-missing** If the tag is missing

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

(End definition for mc-tag-missing. This function is documented on page 18.)

**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.\}
11 \msg_new:nnn { tag } { mc-label-unknown } { label-#1-unknown-or-has-been-already-used.\}
```

(End definition for mc-label-unknown. This function is documented on page 18.)

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

```
\msg_new:nnn { tag } { mc-used-twice } { mc-#1-has-been-already-used }
```

(End definition for mc-used-twice. This function is documented on page 18.)

**mc-not-open** This is issued if a `\tag_mc_end:` is issued wrongly, wrong coding.

```
\msg_new:nnn { tag } { mc-not-open } { there-is-no-mc-to-end-at-#1 }
```

(End definition for mc-not-open. This function is documented on page 18.)
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 definition for mc-pushed and mc-popped. These functions are documented on page 18.)

mc-popped

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=\_tag_mc_key_tag_tl}
  {no-MC-open,-current-abcnt=\_tag_get_mc_abs_cnt:}"
}
(End definition for mc-current. This function is documented on page 18.)

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 definition for struct-unknown. This function is documented on page ??.)

struct-no-objnum  Should not happen ...
\msg_new:nnn { tag } {struct-no-objnum} { objnum-missing-for-structure-#1 }
(End definition for struct-no-objnum. This function is documented on page 18.)

struct-faulty-nesting  This indicates that there is somewhere one \tag\_struct\_end: too much. This should be normally an error.
\msg_new:nnn { tag } {struct-faulty-nesting} { there is no open structure on the stack }
(End definition for struct-faulty-nesting. This function is documented on page 18.)

struct-missing-tag  A structure must have a tag.
\msg_new:nnn { tag } {struct-missing-tag} { a-structure-must-have-a-tag! }
(End definition for struct-missing-tag. This function is documented on page 18.)

struct-used-twice
\msg_new:nnn { tag } {struct-used-twice} { structure-with-label-#1 has already been used}
(End definition for struct-used-twice. This function is documented on page 18.)

struct-label-unknown  label is unknown, typically needs a rerun.
\msg_new:nnn { tag } {struct-label-unknown} { structure-with-label-#1 is unknown-rerun}
(End definition for struct-label-unknown. This function is documented on page 18.)
**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 definition for struct-show-closing. This function is documented on page 18.)

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

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

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

### 3.3 Attributes

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

**attr-unknown**

```
\msg_new:nnn { tag } {attr-unknown} { attribute-#1-is-unknown}
```

(End definition for attr-unknown. This function is documented on page 18.)

### 3.4 Roles

**role-missing**  Warning message if either the tag or the role is missing

```
\msg_new:nnn { tag } {role-missing} { tag-#1-has-no-role-assigned }
```

```
\msg_new:nnn { tag } {role-unknown} { role-#1-is-not-known }
```

```
\msg_new:nnn { tag } {role-unknown-tag} { tag-#1-is-not-known }
```

(End definition for role-missing, role-unknown, and role-unknown-tag. These functions are documented on page 18.)

**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}
{ The-rule-between-parent-’#1’-\and-child-’#2’-is-’#3’}
```

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

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

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

(End definition for role-parent-child. This function is documented on page ??.)
3.5 Miscellaneous

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

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

(End definition for `tree-mcid-index-wrong`. This function is documented on page 19.)

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

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

(End definition for `sys-no-interwordspace`. This function is documented on page 19.)

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

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

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

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

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

(End definition for `para-hook-count-wrong`. This function is documented on page 19.)

4 Retrieving data

**\tag_get:n** This retrieves some data. This is a generic command to retrieve data. Currently the only sensible values for the argument are `mc_tag`, `struct_tag` and `struct_num`.

```latex
(base)\cs_new:Npn \tag_get:n #1 \{ \use:c {\__tag_get_data_#1: } \}
```

(End definition for `\tag_get:n`. This function is documented on page 16.)
5 User conditionals

\tag_if_active_p:TF
\tag_if_active:TF

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

6 Internal checks

These are checks used in various places in the code.

6.1 checks for active tagging

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.
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.

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

(End definition for \__tag_check_structure_has_tag:n.)

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

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

(End definition for \__tag_check_structure_tag:N.)

\__tag_check_info_closing_struct:n
This info message is issued at a closing structure, the use should be guarded by log-level.

```latex
\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}}}
```

(End definition for \__tag_check_info_closing_struct:n.)

\__tag_check_no_open_struct:
This checks if there is an open structure. It should be used when trying to close a structure. It errors if false.

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

(End definition for \__tag_check_no_open_struct.)
\__tag_check_struct_used:n
This checks if a stashed structure has already been used.
\cs_new_protected:Npn \__tag_check_struct_used:n #1 %#1 label
\prop_get:cnNT
\{g__tag_struct__\__tag_ref_value:enn\{tagpdfstruct-#1\}\{tagstruct\}\{unknown\}_prop\}
\l_tmpa_tl
\{\l_tmpa_tl
\msg_warning:nnn \{ tag \} \{struct-used-twice\} \{#1\}
\}

(End definition for \__tag_check_struct_used:n.)

\__tag_check_add_tag_role:nn
This check is used when defining a new role mapping.
\cs_new_protected:Npn \__tag_check_add_tag_role:nn #1 #2 %#1 tag, #2 role
\tl_if_empty:nTF {#2}
\{\l_tmpa_tl
\msg_error:nnn \{ tag \} \{role-missing\} \{#1\}
\}
\prop_get:NnNTF \g__tag_role_tags_NS_prop \{#2\} \l_tmpa_tl
\{\int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
\{\msg_info:nnnn \{ tag \} \{role-tag\} \{#1\} \{#2\}
\}
\}
\msg_error:nnn \{ tag \} \{role-unknown\} \{#2\}
\}

(End definition for \__tag_check_add_tag_role:nn.)

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

Similar with a namespace
\cs_new_protected:Npn \__tag_check_add_tag_role:nnn #1 #2 #3 %#1 tag/NS, #2 role #3 namespace
\tl_if_empty:nTF {#2}
\{\l_tmpa_tl
\msg_error:nnn \{ tag \} \{role-missing\} \{#1\}
\}
\prop_get:cnNnTF \{ g__tag_role_NS_#3_prop \} \{#2\} \l_tmpa_tl
\{\int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
\{\msg_info:nnnn \{ tag \} \{role-tag\} \{#1\} \{#2/#3\}
\}

25
6.4 Check related to mc-chunks

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

\__tag_check_mc_if_nested:
\__tag_check_mc_if_open:

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.

\__tag_check_mc_tag:N

This checks if the mc has a (known) tag.
\prop_if_in:NoF \g__tag_role_tags_NS_prop {#1}
{
\msg_warning:nnx { tag } {role-unknown-tag} {#1}
}
\g__tag_check_mc_used_intarray
\_tag_check_init_mc_used:
This variable holds the list of used mc numbers. Everytime we store a mc-number we will add one the relevant array index If everything is right at the end there should be only 1 until the max count of the mcid. 2 indicates that one mcid was used twice, 0 that we lost one. In engines other than luatex the total number of all intarray entries are restricted so we use only a rather small value of 65536, and we initialize the array only at first used, guarded by the log-level. This check is probably only needed for debugging. TODO does this really make sense to check? When can it happen??
\cs_new_protected:Npn \_tag_check_init_mc_used:
{
\intarray_new:Nn \g__tag_check_mc_used_intarray { 65536 }
\cs_gset_eq:NN \_tag_check_init_mc_used: \prg_do_nothing:
}
(End definition for \g__tag_check_mc_used_intarray and \_tag_check_init_mc_used:.)
\_tag_check_mc_used:n
This checks if a mc is used twice.
\cs_new_protected:Npn \_tag_check_mc_used:n #1 %#1 mcid abscnt
{
\int_compare:nNnT {\l__tag_loglevel_int} > { 2 }
{
\_tag_check_init_mc_used:
\intarray_gset:Nnn \g__tag_check_mc_used_intarray {#1}
{ \intarray_item:Nn \g__tag_check_mc_used_intarray {#1} + 1 }
\int_compare:nNnT
{\intarray_item:Nn \g__tag_check_mc_used_intarray {#1}}
{ 1 }
{
\msg_warning:nnn { tag } {mc-used-twice} {#1}
}
}
(End definition for \_tag_check_mc_used:n.)
\_tag_check_show_MCID_by_page:
This allows to show the mc on a page. Currently unused.
\cs_new_protected:Npn \_tag_check_show_MCID_by_page:
{\tl_set:Nx \l__tag_tmpa_tl
{\_tag_ref_value_lastpage:nn
{abspage}
\_{\text{End definition for}} \textbackslash \_\text{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 \texttt{\textbackslash tag\_mc\_begin:n} (tmb) and \texttt{\textbackslash 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 \seq_if_eq:NNTF doesn’t exist we use the tl-test.

\prg_new_conditional:Npn \_tag_check_if_mc_in_galley: { T,F,TF } { \tl_if_eq:NNTF \l__tag_mc_firstmarks_seq \l__tag_mc_botmarks_seq { \prg_return_false: } { \prg_return_true: } }

(End definition for \_tag_check_if_mc_in_galley:TF)

\__tag_check_if_mc_tmb_missing_p:
\__tag_check_if_mc_tmb_missing:TF
This checks if a extra top mark (“extra-tmb”) is needed. According to the analysis this the case if the firstmarks start with e- or b+. Like above we assume that the marks content is already in the seq’s.

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

(End definition for \_tag_check_if_mc_tmb_missing:TF)

\__tag_check_if_mc_tme_missing_p:
\__tag_check_if_mc_tme_missing:TF
This checks if a extra bottom mark (“extra-tme”) is needed. According to the analysis this the case if the botmarks starts with b+. Like above we assume that the marks content is already in the seq’s.

\prg_new_conditional:Npn \_tag_check_if_mc_tme_missing: { T,F,TF } { \str_if_eq:eeTF {\seq_item:Nn \l__tag_mc_botmarks_seq {1}}{b+} { \prg_return_true: } { \prg_return_false: } }

(End definition for \_tag_check_if_mc_tme_missing:TF)

\langle/package\rangle
\langle∗debug∗⟩
Code for tagpdf-debug. This will probably change over time. At first something for the mc commands.

\msg_new:nnn { tag / debug } {mc-begin} { MC~begin~#1~with~options:\tl_to_str:n{#2}~[\msg_line_context:] }
\msg_new:nnn { tag / debug } {mc-end} { MC~end~#1~[\msg_line_context:] }
\cs_new_protected:Npn \__tag_debug_mc_begin_insert:n #1 { \int_compare:nNnT { \l__tag_loglevel_int } > {0} { \msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 } }
\msg_note:nnnn { tag / debug } {mc-begin} {inserted} { #1 } }

29
And now something for the structures
\msg_new:nnn { tag / debug } {struct-begin}
{ Struct~\tag_get:n{struct_num}~begin~#1~with~options:~\tl_to_str:n{#2}~\{\msg_line_context:\} }
\msg_new:nnn { tag / debug } {struct-end}
{ Struct-end~#1~\\{\msg_line_context:\} }
\cs_new_protected:Npn \__tag_debug_struct_begin_insert:n #1
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
  { \msg_note:nnnn { tag / debug } {struct-begin} {inserted} { #1 }
    \seq_log:N \g__tag_struct_tag_stack_seq
  }
}
\cs_new_protected:Npn \__tag_debug_struct_begin_ignore:n #1
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
  { \msg_note:nnnn { tag / debug } {struct-begin} {ignored} { #1 }
  }
}
\cs_new_protected:Npn \__tag_debug_struct_end_insert:
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
  { \msg_note:nnn { tag / debug } {struct-end} {inserted}
    \seq_log:N \g__tag_struct_tag_stack_seq
  }
}
\cs_new_protected:Npn \__tag_debug_struct_end_ignore:
{ \int_compare:nNnT { \l__tag_loglevel_int } > {0}
  { \msg_note:nnn { tag / debug } {struct-end} {ignored}
  }
}
\cs_new_protected:Npn \__tag_debug_struct_end_ignore:
{
\int_compare:nNnT { \l__tag_loglevel_int } > {0}
{
\msg_note:nnn { tag / debug } {struct-end} {ignored}
}
}
(/debug)
Part II

The \texttt{tagpdf-user} module

Code related to \LaTeX\,\textsc{2e} user commands and document commands

Part of the tagpdf package

1 Setup commands

\begin{verbatim}
\tagpdfsetup \tagpdfsetup{\langle key val list\rangle}
\end{verbatim}

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).

\begin{verbatim}
activate\_\textsc{setup-key}
\end{verbatim}

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

\begin{verbatim}
\tagpdfifluatexTF
\tagpdfifpdftexT
\end{verbatim}

small wrappers around engine tests. This functions should not be used and will be removed in one of the next versions.

2 Commands related to mc-chunks

\begin{verbatim}
\tagmcbegin \tagmcbegin {\langle key-val\rangle}
\tagmcend \tagmcend
\tagmcuse {\langle label\rangle}
\end{verbatim}

These are wrappers around \texttt{\tag\_mc\_begin:}n, \texttt{\tag\_mc\_end}: and \texttt{\tag\_mc\_use:}n. The commands and their argument are documented in the \texttt{tagpdf-mc} module. In difference to the expl3 commands, \texttt{\tagmcbegin} issues also an \texttt{\ignorespaces}, and \texttt{\tagmcend} will issue in horizontal mode an \texttt{\unskip}.

\begin{verbatim}
\tagmcifinTF \tagmcifin {\langle true code\rangle}{\langle false code\rangle}
\end{verbatim}

This is a wrapper around \texttt{\tag\_mc\_if\_in:TF} and tests if an mc is open or not. It is mostly of importance for pdflatex as lualatex doesn’t mind much if a mc tag is not correctly closed. Unlike the expl3 command it is not expandable.

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

\tagstructbegin \tagstructbegin \{\textlangle key-val\rangle\}\tagstructend \tagstructend \tagstructuse\{\textlangle 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 \{\textlangle 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.

\textbf{mc-data}(\textlangle show-key\rangle) mc-data = \langle 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.

\textbf{mc-current}(\textlangle 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.

\textbf{mc-marks}(\textlangle show-key\rangle) mc-marks = show|use

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

\textbf{struct-stack}(\textlangle show-key\rangle) struct-stack = log|show

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

5 Extension commands

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

The commands and keys should be view as experimental!

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

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

5.2 Paratagging

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

\paratagging\,(setup-key) \paratagging = true|false
\paratagging-show\,(setup-key) \paratagging-show = true|false

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

\tagpdfparaOn \tagpdfparaOff

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

\tagpdfsuppressmarks

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

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

5.3 Header and footer

Header and footer are automatically excluded from tagging. This can be disabled with the following key. If some real content is in the header and footer, tagging must be restarted there explicitly. The key accepts the values true which surrounds the header with an artifact mc-chunk, false which disables the automatic tagging, and pagination which additionally adds an artifact structure with an pagination attribute.

\exclude-header-footer\,(setup-key) \exclude-header-footer = true|false|pagination
5.4 Link tagging

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

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

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

6 User commands and extensions of document commands

\tagpdfsetup
\tagmcbegin
\tagmcend
\tagmcuse

7 Setup and preamble commands

(End definition for \tagpdfsetup. This function is documented on page 32.)

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

\tagstructbegin\tagstructend\tagstructuse
These are structure related user commands. There are direct wrapper around the expl3
variants.

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

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

(End definition for \ShowTagging. This function is documented on page 33.)

mc-current_{show-key} This shows some info about the current mc-chunk. It works in generic and lua-mode.

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

(End definition for mc-data (show-key). This function is documented on page 33.)
(End definition for \mc-current (show-key). This function is documented on page 33.)

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

\keys_define:nn { __tag / show }
{ }
\mc-marks .choice: ,
\mc-marks / show .code:n =
{ 
  \__tag_mc_get_marks:
  \__tag_check_if_mc_in_galley:TF
  { 
    \io_term:n {Marks-from-this-page:-}
  } 
  { 
    \io_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
{ }
The following commands and code parts are not core command of tagpdf. The either provide work arounds for missing functionality elsewhere, or do a first step to apply tagpdf commands to document commands. This part should be regularly revisited to check if the code should go to a better place or can be improved.

11 Commands to extend document commands

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

11.1 Document structure
\hook_gput_code:nnn{tagpdf/finish/before}{tagpdf}{\tagstructend}

\keys_define:nnn { __tag } { setup } { activate .code:n = activate .default:n = Document }

activate .code:n = activate .default:n = Document

(End definition for \__tag_add_document_structure:n and activate (setup-key). This function is documented on page 32.)

11.2 Structure destinations

In TeXlive 2022 pdftex and luatex will offer support for structure destinations. The pdfmanagement has already backend support. We activate them if the prerequisites are there: structures should be activated, the code in the pdfmanagement must be there. 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_all:nT } { \__tag_fakespace: }

(End definition for \__tag_add_document_structure:n and activate (setup-key). This function is documented on page 32.)

11.3 Fake space

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

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

(End definition for \pdffakespace. This function is documented on page 34.)
11.4 Paratagging

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

At first some variables.

\bool_new:N \l__tag_para_bool
\bool_new:N \l__tag_para_show_bool
\int_new:N \g__tag_para_begin_int
\int_new:N \g__tag_para_end_int
\tl_new:N \l__tag_para_tag_default_tl
\tl_new:N \l__tag_para_tag_tl
\tl_set:Nn \l__tag_para_tag_default_tl { P }
\tl_new:N \l__tag_para_tag_tl
\tl_set:Nn \l__tag_para_tag_tl { \l__tag_para_tag_default_tl }

(End definition for \l__tag_para_bool and others.)

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

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

(End definition for paratagging (setup-key) and paratagging-show (setup-key). These functions are documented on page 34.)

This fills the para hooks with the needed code.

\AddToHook{para/begin} {
  \bool_if:NT \l__tag_para_bool {
    \int_gincr:N \g__tag_para_begin_int
    \tag_struct_begin:n {tag=\l__tag_para_tag_tl}
    \bool_if:NT \l__tag_para_show_bool {
      \tag_mc_begin:n{artifact}
      \llap{\color_select:n{red}\tiny\int_use:N\g__tag_para_begin_int}
      \tag_mc_end:
    }
    \tag_mc_begin:n {}
  }
}

\AddToHook{para/end} {
  \bool_if:NT \l__tag_para_bool {
    \int_gincr:N \g__tag_para_end_int
    \tag_mc_end:
    \bool_if:NT \l__tag_para_show_bool {
      \tag_mc_begin:n{artifact}
      \rlap{\color_select:n{red}\tiny \int_use:N\g__tag_para_end_int}
    }
  }
}
In generic mode we need the additional code from the ptagging tests.

\AddToHook{begindocument/before}{
    \bool_if:NF \g__tag_mode_lua_bool {
        \cs_if_exist:NT \@kernel@before@footins {
            \tl_put_right:Nn \@kernel@before@footins { \__tag_add_missing_mcs_to_stream:Nn \footins {footnote} } 
        }
        \tl_put_right:Nn \@kernel@before@cclv {
            \__tag_check_typeout_v:n {====>~In~\token_to_str:N \@makecol\c_space_tl\the\c@page} 
            \__tag_add_missing_mcs_to_stream:Nn \@cclv {main} 
        }
        \tl_put_right:Nn \mult@ptagging@hook {
            \__tag_check_typeout_v:n {====>~In~\string\page@sofar} 
            \process@cols\mult@firstbox {
                \__tag_add_missing_mcs_to_stream:Nn \count@ {multicol} 
                \__tag_add_missing_mcs_to_stream:Nn \mult@rightbox {multicol} 
            }
        }
    }
}

\AddToHook{enddocument/info}{
    \int_compare:nNnF {\g__tag_para_begin_int}={\g__tag_para_end_int} {
        \msg_error:nnxx {tag} {para-hook-count-wrong} 
        \int_use:N \g__tag_para_begin_int 
        \int_use:N \g__tag_para_end_int 
    }
}

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

\tagpdfparaOn\tagpdfparaOff

(End definition for \tagpdfparaOn and \tagpdfparaOff. These functions are documented on page 34.)
This command allows to suppress the creation of the marks. It takes an argument which should normally be one of the mc-commands, puts a group around it and suppress the marks creation in this group. This command should be used if the begin and end command are at different boxing levels. E.g.

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

\NewDocumentCommand\tagpdfsuppressmarks{m}
{{\use:c{\_tag_mc_disable_marks:} #1}}

(End definition for \tagpdfsuppressmarks. This function is documented on page 34.)

11.5 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:
  }
  \tag_mc_begin:n {artifact}
}

\cs_new_protected:Npn \__tag_exclude_headfoot_end:
{
This version allows to use an Artifact structure

\__tag\_attr\_new\_entry:nn {\__tag/attr/pagination}{/O/Artifact/Type/Pagination}
\cs\new\protected:Npn \__tag\_exclude\_struct\_headfoot\_begin:n #1
{
  \bool\set\false:N \l__tag\_para\_bool
  \bool_if:NTF \g__tag\_mode\_lua\_bool
  {
    \tag\_mc\_end\_push:
  }
  \bool_gset_eq:NN \g__tag\_saved\_in\_mc\_bool \g__tag\_in\_mc\_bool
  {\bool_gset\false:N \g__tag\_in\_mc\_bool
  }
  \tag\_struct\_begin:n{tag=Artifact,attribute-class=\__tag\_attr/#1}
  \tag\_mc\_begin:n {artifact=#1}
}
\cs\new\protected:Npn \__tag\_exclude\_struct\_headfoot\_end:
{
  \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 }
{
  exclude-header-footer .choice:,
  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\_begin:
    \cs\set_eq:NN \__tag\_hook\_kernel\_after\_foot: \__tag\_exclude\_headfoot\_end:
  },
  exclude-header-footer / pagination .code:n =
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:
},
exclude-header-footer .default:n = true,
exclude-header-footer .initial:n = true

(End definition for exclude-header-footer (setup-key). This function is documented on page 34.)

11.6 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_mc_end:n { tag=Link }
    \tag_mc_begin:n { tag=Link }
    \pdfannot_dict_put:nnx { link/URI } { StructParent }
    { \tag_struct_parent_int: }
}

\hook_gput_code:nnn
{pdfannot/link/URI/after}
{tagpdf}
{
    \tag_struct_insert_annot:xx {\pdfannot_link_ref_last:}{\tag_struct_parent_int:}
    \tag_mc_end:
    \tag_mc_end:
    \tag_mc_begin_pop:n{} }
% "alternative descriptions " for PAX3. How to get better text here??

</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:

\cs_new_protected:Npn \__tag_tree_final_checks:
\int_compare:nNnF {\seq_count:N\g__tag_struct_stack_seq}=1
\msg_warning:n {tag}{tree-struct-still-open}
\int_step_inline:nnn{2}{\seq_count:N\g__tag_struct_stack_seq}{\tag_struct_end:}

(End definition for \__tag_tree_final_checks:)
1.2 Catalog: MarkInfo and StructTreeRoot

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

This is the object for the root object, the StructTreeRoot

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

```
30 \hook_gput_code:nnn{shipout/lastpage}{tagpdf}{
31 \bool_if:NT \g__tag_active_tree_bool
32 { \pdfmanagement_add:nnn { Catalog / MarkInfo } { Marked } { true }
33 \pdfmanagement_add:nnx
34 { Catalog }
35 { StructTreeRoot }
36 { \pdf_object_ref:n { __tag/struct/0 } }
37 }
38 }
```

1.3 Writing structure elements

The following commands are needed to write out the structure.

This writes out the root object.

```
41 \pdf_version_compare:NnTF < {2.0}
42 { \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
43 { \_tag_prop_gput:cnx
44 \{ g__tag_struct_0_prop \}
45 \{ ParentTree \}
46 \{ \pdf_object_ref:n { __tag/tree/parenttree } \}
47 \_tag_prop_gput:cnx
48 \{ g__tag_struct_0_prop \}
49 \{ RoleMap \}
50 \{ \pdf_object_ref:n { __tag/tree/rolemap } \}
51 \_tag_struct_write_obj:n { 0 }
52 }
53 }
```

no RoleMap in pdf 2.0

```
56 { \cs_new_protected:Npn \__tag_tree_write_structtreeroot:
57 { \_tag_prop_gput:cnx
58 \{ g__tag_struct_0_prop \}
59 \{ ParentTree \}
60 \{ \pdf_object_ref:n { __tag/tree/parenttree } \}
61 \_tag_struct_write_obj:n { 0 }
62 }
63 }
```

(End definition for \_tag_tree_write_structtreeroot:)
\__tag_tree_write_structelements:  This writes out the other struct elems, the absolute number is in the counter.
\cs_new_protected:Npn \__tag_tree_write_structelements:
  {\int_step_inline:nnnn {1}{1}{\c@g__tag_struct_abs_int}
  { \__tag_struct_write_obj:n { ##1 }}}
(End definition for \__tag_tree_write_structelements:.)

1.4 ParentTree
\pdf_object_new:n { \__tag/tree/parenttree }
(End definition for \__tag/tree/parenttree.)
The ParentTree maps numbers to objects or (if the number represents a page) to arrays of objects. The numbers refer to two 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.
\newcounter { g__tag_parenttree_obj_int }
\hook_gput_code:nnn{begindocument}{tagpdf}{\int_gset:Nn \c@g__tag_parenttree_obj_int { \__tag_ref_value_lastpage:nn{abspage}{100} }}
(End definition for \c@g__tag_parenttree_obj_int.)
We store the number/object references in a tl-var. If more structure is needed one could switch to a seq.
\tl_new:N \g__tag_parenttree_objr_tl
(End definition for \g__tag_parenttree_objr_tl.)
\__tag_parenttree_add_objr:nn This command stores a StructParent number and a objref into the tl var. This is only for objects like annotations, pages are handled elsewhere.
\cs_new_protected:Npn \__tag_parenttree_add_objr:nn #1 #2 % #1 StructParent number, #2 objref
  { \tl_gput_right:Nx \g__tag_parenttree_objr_tl { #1 \c_space_tl #2 ^^J } }
(End definition for \__tag_parenttree_add_objr:nn.)
\l__tag_parenttree_content_tl \ A tl-var which will get the page related parenttree content.
\tl_new:N \l__tag_parenttree_content_tl
(End definition for \l__tag_parenttree_content_tl.)

\__tag_tree_fill_parenttree: \ This is the main command to assemble the page related entries of the parent tree. It
wanders through the pages and the mcid numbers and collects all mcid of one page.
\cs_new_protected:Npn \__tag_tree_fill_parenttree:
\prop_clear:N \l__tag_tmpa_prop
\int_step_inline:nnnn{0}{1}{\prop_count:N \l__tag_tmpa_prop}
\prop_get:NnNTF \l__tag_tmpa_prop {###1} \l__tag_tmpa_tl
\msg_warning:nn {tag} {tree-mcid-index-wrong}
\tl_put_right:Nn \l__tag_parenttree_content_tl
{[\c_space_tl ]
\_tag\_lua\_fill\_parenttree:  This is a special variant for luatex. lua mode must/can do it differently.

\cs\_new\_protected:Npn \_tag\_lua\_fill\_parenttree:
\{ 
  \tl\_set:Nn \l\_tag\_parenttree\_content\_tl 
  \{ 
    \lua\_now:e 
    \begin{verbatim} 
    \ltx\_tag\_func\_output\_parenttree 
    \{ 
      \int\_use:N\g\_shipout\_readonly\_int 
    \} 
    \end{verbatim} 
  \} 
\}

(End definition for \_tag\_lua\_fill\_parenttree:)

\_tag\_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\_write\_parenttree:
\{ 
  \bool\_if:NTF \g\_tag\_mode\_lua\_bool 
  \{ 
    \_tag\_lua\_fill\_parenttree: 
  \} 
  \{ 
    \_tag\_fill\_parenttree: 
  \} 
  \tl\_put\_right:NV \l\_tag\_parenttree\_content\_tl \g\_tag\_parenttree\_objr\_tl 
  \pdf\_object\_write:nnx \{ \_tag\_tree\_parenttree \} \{ \dict \} 
  \{ 
    /Nums\c\_space\_tl \l\_tag\_parenttree\_content\_tl 
  \} 
\}

(End definition for \_tag\_write\_parenttree:)

1.5 Rolemap dictionary

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

\_tag\_tree\_rolemap  At first we reserve again an object.

\pdf\_version\_compare:NnT < \{2.0\} 
\{ 
  \pdf\_object\_new:n \{ \_tag\_tree\_rolemap \} 
\}

51
This writes out the rolemap, basically it simply pushes out the dictionary which has been filled in the role module.

```latex
\pdf_version_compare:NnTF < {2.0}
{
\cs_new_protected:Npn \_tag_tree_write_rolemap:
{
\prop_map_inline:Nn \g__tag_role_rolemap_prop
{\pdfdict_gput:nnx {\g__tag_role/RoleMap_dict}{##1}{\pdf_name_from_unicode_e:n{##2}}}
\pdf_object_write:nnx { __tag/tree/rolemap }{dict}
{\pdfdict_use:n {\g__tag_role/RoleMap_dict}}
}
}
{
\cs_new_protected:Npn \_tag_tree_write_rolemap:{}
}
}
```

(End definition for \_tag_tree_write_rolemap.)

### 1.6 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.

```latex
\cs_new_protected:Npn \_tag_tree_write_classmap:
{
\tl_clear:N \l__tag_tmpa_tl
\seq_gremove_duplicates:N \g__tag_attr_class_used_seq
\seq_set_map:NNn \l__tag_tmpa_seq \g__tag_attr_class_used_seq
{##1\c_space_tl
\prop_item:Nn \g__tag_attr_entries_prop
{##1}
\seq_use:Nn \l__tag_tmpa_seq { \iow_newline: }
}
\tl_if_empty:NF \l__tag_tmpa_tl
{
\tl_set:Nx \l__tag_tmpa_tl
{\seq_use:Nn \l__tag_tmpa_seq { \tl_if_empty:NF \l__tag_tmpa_tl
}
```

(End definition for \_tag_tree_write_classmap.)
1.7 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

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

(End definition for \__tag_tree_write_namespaces:)
1.8 Finishing the structure

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

\__tag_finish_structure:

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

(End definition for \__tag_finish_structure:)

1.9 StructParents entry for Page

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

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

/package
Part IV

The tagpdf-mc-shared module

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

Part of the tagpdf package

1 Public Commands

\tag_mc_begin:n \tag_mc_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:

Rev: 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_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.
2 Public keys

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

**tag\_mc\_key\_key**

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

**artifact\_mc\_key\_key**

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

**raw\_mc\_key\_key**

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

**alt\_mc\_key\_key**

This key inserts an /Alt value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. The value will be expanded first once.

**actualtext\_mc\_key\_key**

This key inserts an /ActualText value in the property dictionary of the BDC operator. The value is handled as verbatim string, commands are not expanded. The value will be expanded first once.

**label\_mc\_key\_key**

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

**stash\_mc\_key\_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

```latex
{part of tagpdf - code related to marking chunks -
code shared by generic and luamode }
```

56
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{\guf_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.

\begin{verbatim}
g__tag_MCID_abs_int
  \texttt{(shared)}
  \texttt{\newcounter{g__tag_MCID_abs_int}}
\end{verbatim}

\begin{verbatim}
\__tag_get_mc_abs_cnt:
  \texttt{A (expandable) function to get the current value of the cnt.}
  \texttt{\cs_new:Npn \__tag_get_mc_abs_cnt: { \int_use:N \c@g__tag_MCID_abs_int }}
\end{verbatim}

\begin{verbatim}
g__tag_MCID_tmp_bypage_int
  \texttt{The following hold the temporary by page number assigned to a mc. It must be defined in the shared code to avoid problems with labels.}
  \texttt{\int_new:N \g__tag_MCID_tmp_bypage_int}
\end{verbatim}

\begin{verbatim}
g__tag_in_mc_bool
  \texttt{This booleans record if a mc is open, to test nesting.}
  \texttt{\bool_new:N \g__tag_in_mc_bool}
\end{verbatim}

\begin{verbatim}
g__tag_mc_parenttree_prop
  \texttt{For every chunk we need to know the structure it is in, to record this in the parent tree.}
  \texttt{We store this in a property.}
  \texttt{key: absolute number of the mc (tagmcabs)}
  \texttt{value: the structure number the mc is in}
  \texttt{\__tag_prop_new:N \g__tag_mc_parenttree_prop}
\end{verbatim}

\begin{verbatim}
g__tag_mc_stack_seq
  \texttt{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:}
  \texttt{\seq_new:N \g__tag_mc_stack_seq}
\end{verbatim}

\begin{verbatim}
l__tag_mc_artifact_type_tl
  \texttt{Artifacts can have various types like Pagination or Layout. This stored in this variable.}
  \texttt{\tl_new:N \l__tag_mc_artifact_type_tl}
\end{verbatim}

\begin{verbatim}
l__tag_mc_key_stash_bool
  \texttt{This booleans store the stash and artifact status of the mc-chunk.}
l__tag_mc_artifact_bool
  \texttt{(End definition for l__tag_mc_key_stash_bool and l__tag_mc_artifact_bool.)}
\end{verbatim}
Variables used by the keys. \l_@@_mc_key_properties_tl will collect a number of values. TODO: should this be a pdffict now?

\l_new:N \l__tag_mc_key_tag_tl
\l_new:N \g__tag_mc_key_tag_tl
\l_new:N \l__tag_mc_key_label_tl
\l_new:N \l__tag_mc_key_properties_tl

(End definition for \l__tag_mc_key_tag_tl and others.)

### 3.2 Functions

The commands labels a mc-chunk. It is used if the user explicitly labels the mc-chunk with the label key. The argument is the value provided by the user. It stores the attributes

- `tagabpage`: the absolute page, \g_shipout_readonly_int.
- `tagmcabs`: the absolute mc-counter \c@g_@@_MCID_abs_int.
- `tagmcid`: the ID of the chunk on the page \g_@@_MCID_tmp_bypage_int, this typically settles down after a second compilation. The reference command is defined in tagpdf.dtx and is based on l3ref.

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

(End definition for \__tag_mc_handle_mc_label:n.)

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

(End definition for \__tag_mc_set_label_used:n.)

\tag_mc_use:n

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

TODO: is testing for struct the right test?

\cs_set_protected:Npn \tag_mc_use:n #1 %#1: label name
\__tag_check_if_active_struct:T
\__tag_ref_value:nnn{tagpdf-#1}{tagmcabs}{}``
\tl_set:Nx \l__tag_tmpa_tl { \__tag_ref_value:nnn{tagpdf-#1}{tagmcabs}{} }
\tl_if_empty:NTF\l__tag_tmpa_tl
\msg_warning:nnn {tag} {mc-label-unknown} {#1}
\}
\}

(End definition for \__tag_mc_set_label_used:n.)

58
\cs_if_free:cTF { g__tag_mc_label\_tl_to_str:n{#1}_used_tl }
{
  \__tag_mc_handle_stash:x { \l__tag_tmpa_tl
  \__tag_mc_set_label_used:n {#1}
}
{
  \msg_warning:nnn {tag}{mc-used-twice}{#1}
}
}

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

\cs_set_protected:Npn \tag_mc_artifact_group_begin:n #1 {}
{\tag_stop_group_begin:
\tag_mc_begin:n {artifact=#1}
\tag_mc_end:
\tag_mc_end_push:
\tag_mc_begin_pop:n{}}

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

\cs_set_protected:Npn \tag_mc_artifact_group_end: {}
{\tag_stop_group_end:
\tag_mc_end:
\tag_mc_begin_pop:n{}}

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

(End definition for \tag_mc_use:n. This function is documented on page 55.)

\tag_mc_artifact_group_begin:n

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.

\tag_mc_artifact_group_end:

(End definition for \tag_mc_artifact_group_begin:n and \tag_mc_artifact_group_end: These functions are documented on page 55.)

\tag_mc_end_push:

\tag_mc_begin_pop:n

(End definition for \tag_mc_artifact_group_begin:n and \tag_mc_artifact_group_end: These functions are documented on page 55.)
\seq_gpush:Nn \g__tag_mc_stack_seq {-1} \__tag_check_mc_pushed_popped:nn { pushed }{-1} 
\__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing} 
\cs_set_protected:Npn \tag_mc_begin_pop:n #1 
\__tag_check_if_active_mc:T \seq_gpop:NNTF \g__tag_mc_stack_seq \l__tag_tmpa_tl \tl_if_eq:NnTF \l__tag_tmpa_tl {-1} \__tag_check_mc_pushed_popped:nn {popped}{-1} \__tag_check_mc_pushed_popped:nn {popped}{\l__tag_tmpa_tl} \tag_mc_begin:n {tag=\l__tag_tmpa_tl,#1} 
\__tag_check_mc_pushed_popped:nn {popped}{empty~stack,~nothing} 
(End definition for \tag_mc_end_push: and \tag_mc_begin_pop:n. These functions are documented on page 55.)

3.3 Keys

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

\keys_define:nn { __tag / mc } 
\stash\mc-key \__artifact-bool \__artifact-type 
\tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header } 
\tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer } 

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

\stash\mc-key \__artifact-bool \__artifact-type 
\tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Header } 
\tl_set:Nn \l__tag_mc_artifact_type_tl { Pagination/Subtype/Footer }
__artifact-type / layout .code:n = \tl_set:Nn \l__tag_mc_artifact_type_tl { Layout }

__artifact-type / page .code:n = \tl_set:Nn \l__tag_mc_artifact_type_tl { Page }

__artifact-type / background .code:n = \tl_set:Nn \l__tag_mc_artifact_type_tl { Background }

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

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

(End definition for stash (mc-key), __artifact-bool, and __artifact-type. This function is documented on page 56.)
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

\g__tag_MCID_byabspage_prop
This property will hold the current maximum on a page it will contain key-value of type \langle \text{abspagenum} \rangle = \langle \text{max mcid} \rangle

\l__tag_mc_ref_abspage_tl
We need a ref-label system to ensure that the MCID cnt restarts at 0 on a new page. This will be used to store the tagabspage attribute retrieved from a label.

\l__tag_mc_tmpa_tl
Temporary variable

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

```latex
\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
```

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.

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

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).

```latex
\cs_new_protected:Npn \__tag_mc_begin_marks:nn #1 #2 %#1 tag, #2 label
\cs_generate_variant:Nn \__tag_mc_begin_marks:nn { oo }
\cs_new_protected:Npn \__tag_mc_artifact_begin_marks:n #1 %#1 type
```

```latex
\cs_new_protected:Npn \__tag_mc_end_marks:
\cs_generate_variant:Nn \__tag_mc_end_marks: { oo }
```
\newcommand*{\mcidabs}{\pgfmathparse{#1}}
\newcommand*{\mcstruct}{\pgfmathparse{#1}}
\newcommand*{\mcnum}{\pgfmathparse{#1}}
\newcommand{\__tag mc disable marks}{\cs_set_eq:NN \__tag mc begin marks:nn \use_none:nn \cs_set_eq:NN \__tag mc artifact begin marks:n \use_none:n \cs_set_eq:NN \__tag mc end marks: \prg_do_nothing:}
\newcommand{\__tag mc get marks}{\__tag mc disable marks:
\cs_set_eq:NN \__tag mc begin marks:nn \use_none:nn \cs_set_eq:NN \__tag mc artifact begin marks:n \use_none:n \cs_set_eq:NN \__tag mc end marks: \prg_do_nothing:}
\__tag mc disable 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.
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.

\begin{verbatim}
\cs_new_protected:Npn \__tag_mc_store:nnn #1 #2 #3 %#1 mc-prev, #2 mc-num #3 structure-num
\{
  %\prop_show:N \g__tag_struct_cont_mc_prop
  \prop_get:NnNTF \g__tag_struct_cont_mc_prop {#1} \l__tag_tmpa_tl
  \{
    \prop_gput:Nnx \g__tag_struct_cont_mc_prop {#1}{ \l__tag_tmpa_tl \__tag_struct_mcid_dict:n {#2}}
  \}
  \prop_gput:Nnx \g__tag_struct_cont_mc_prop {#1}{\__tag_struct_mcid_dict:n {#2}}
  \{
    \prop_gput:Nxx \g__tag_mc_parenttree_prop {#2}
    {#3}
  \}
  \prop_gput:Nxx \g__tag_mc_parenttree_prop
\}
\cs_generate_variant:Nn \__tag_mc_store:nnn {xxx}
\end{verbatim}

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.

\begin{verbatim}
\cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream: e.g. main or footnote
\{
  \__tag_check_typeout_v:n {=>~ first~ \seq_use:Nn \l__tag_mc_firstmarks_seq {,~}}
  \__tag_check_typeout_v:n {=>~ bot~ \seq_use:Nn \l__tag_mc_botmarks_seq {,~}}
  \__tag_check_if_mc_tmb_missing:TF
  \{
    \__tag_check_typeout_v:n {=>~ TMB~ missing~ --~ inserted}
    %test if artifact
    \int_compare:nNnTF { \seq_item:cn { g__tag_mc_#1_marks_seq } {3} } = {- \}
    \{
      \tl_set:Nx \l__tag_tmpa_tl \seq_item:cn { g__tag_mc_#1_marks_seq } {4} \}
      \__tag_mc_handle_artifact:N \l__tag_tmpa_tl
    \}
    \{\exp_args:Nx \__tag_mc_bdc_mcid:n \seq_item:cn { g__tag_mc_#1_marks_seq } {4} \}
    \str_if_eq:eeTF \}
\end{verbatim}
\seq_item:cn { g__tag_mc_#1_marks_seq } {5}
\}
\}
{ %store
__tag_mc_store:xxx
{ \seq_item:cn { g__tag_mc_#1_marks_seq } {2} }
\int_eval:n{\c@g__tag_MCID_abs_int} }
{ \seq_item:cn { g__tag_mc_#1_marks_seq } {3} }
}
%
stashed -> warning!!
}
}
\__tag_check_typeout_v:n {=>~ TMB~ ~ not~ ~ missing}
}
}
\cs_new_protected:Npn \__tag_mc_insert_extra_tmb:n #1 % #1 stream, eg. main or footnote
{ \__tag_check_if_mc_tmb_missing:TF
 { \__tag_check_typeout_v:n {=>~ TMB~ ~ not~ ~ missing} \__tag_mc_emc:
 \seq_gset_eq:cN
 { g__tag_mc_#1_marks_seq }
 \l__tag_mc_botmarks_seq
 }
 \{ \__tag_check_typeout_v:n {=>~ TME~ ~ missing~ ~ inserted} \}
}
(End 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 \texttt{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 \texttt{main}
for the main galley, \texttt{footnote} for footnote note text, or \texttt{multicol} for boxes produced for
columns in that environment. Other streams may follow over time.

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

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

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

\begin{Verbatim}
\begin{verbatim}
\texttt{\box_use_drop:N \l__tag_tmpa_box}
\texttt{\vbox_unpack_drop:N \#1}
\end{verbatim}
\end{Verbatim}

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

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

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

\begin{Verbatim}
\begin{verbatim}
\texttt{\cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn \#1 \#2 \{}
\texttt{\__tag_check_if_active_mc:T \{}
\texttt{\__tag_mc_insert_extra_tmb:n \{\#2\} \}}
\texttt{\}}
\end{verbatim}
\end{Verbatim}
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).

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:NNx
\seq_set_from_clist:Nn \l__tag_mc_firstmarks_seq { \tex_splitfirstmarks:D \g__tag_mc_marks }

Some debugging info:

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

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

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

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

\seq_set_eq:NN \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq

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

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

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

\__tag_add_missing_mcs:Nn \l__tag_mc_botmarks_seq \l__tag_mc_firstmarks_seq
\seq_gset_eq:cN {g__tag_mc_#2_marks_seq} \l__tag_mc_botmarks_seq
\__tag_add_missing_mcs_to_stream:Nn

(End definition for \__tag_add_missing_mcs_to_stream:Nn.)
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.

\begin{verbatim}
\prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
\begin{verbatim}
\prg_new_eq_conditional:NNn \tag_mc_if_in: \\__tag_mc_if_in: {p,T,F,TF}
\end{verbatim}
\end{verbatim}

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.

\begin{verbatim}
\cs_set_eq:NN \__tag_mc_bmc:n \pdf_bmc:n
\cs_set_eq:NN \__tag_mc_emc: \pdf_emc:
\cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
\cs_generate_variant:Nn \__tag_mc_bdc:nn {nx}
\end{verbatim}

This create a BDC mark with an /MCID key. Most of the work here is to get the current number value for the MCID: they must be numbered by page starting with 0 and then successively. The first argument is the tag, e.g. P or Span, the second is used to pass more properties. We also define a wrapper around the low-level command as luamode will need something different.

\begin{verbatim}
\cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
\begin{verbatim}
\prop_get:NoNTF \g__tag_MCID_byabspage_prop
\l__tag_mc_ref_abspage_tl
\end{verbatim}
\end{verbatim}

% #1 tag, #2 properties
\begin{verbatim}
\cs_set_eq:NN \__tag_mc_bmc:n \pdf_bmc:n
\cs_set_eq:NN \__tag_mc_emc: \pdf_emc:
\cs_set_eq:NN \__tag_mc_bdc:nn \pdf_bdc:nn
\cs_generate_variant:Nn \__tag_mc_bdc:nn {nx}
\end{verbatim}

(End 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. We also define a wrapper around the low-level command as luamode will need something different.

\begin{verbatim}
\cs_new_protected:Npm \__tag_mc_bdc_mcid:nn #1 #2
\begin{verbatim}
\prop_get:NoNTF \g__tag_MCID_byabspage_prop
\end{verbatim}
\end{verbatim}

(End definition for \__tag_mc_bmc:n, \__tag_mc_emc:, and \__tag_mc_bdc:nn.)

\end{verbatim}
\begin{verbatim}
\l__tag_mc_tmpa_tl
\l__tag_MCID_tmp_bypage_int
\l__tag_mc_ref_abspage_tl
\l__tag_mc_abs_tl
\int_gset:Nn \g__tag_MCID_tmp_bypage_int { \l__tag_mc_tmpa_tl }
\g__tag_MCID_byabspage_prop
{ \l__tag_mc_ref_abspage_tl }
{ \int_eval:n { \l__tag_mc_tmpa_tl +1 } }
\g__tag_MCID_tmp_bypage_int
\g__tag_mc_ref_abspage_prop
{ \l__tag_mc_ref_abspage_tl }
{1}
\__tag_ref_label:en
{ mcid-\int_use:N \c@g__tag_MCID_abs_int}
{ mc}
\__tag_mc_bdc:nx
{ /MCID~\int_eval:n { \g__tag_MCID_tmp_bypage_int }~ \exp_not:n { #2 } }
\__tag_mc_bdc_mcid:n #1
{ \__tag_mc_bdc_mcid:nn {#1} {} }
\cs_new_protected:Npn \__tag_mc_bdc_mcid:nn #1 #2
{ \__tag_mc_bdc_mcid:nn {#1} {#2} }
\cs_generate_variant:Nn \__tag_mc_bdc_mcid:nn {VV}
\__tag_mc_handle_stash:n
\__tag_mc_handle_stash:x
\__tag_check_mc_used:n {#1}
\__tag_struct_kid_mc_gput_right:nn
{ \g__tag_struct_stack_current_tl }
{ \g__tag_struct_stack_current_tl }
{#1}
\prop_gput:Nxx \g__tag_mc_parenttree_prop
{#1}
\end{verbatim}

This is the handler which puts a mc into 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?

\begin{verbatim}
\cs_new_protected:Npm \__tag_mc_handle_stash:n #1 %1 mcidnum
{ \__tag_check_mc_used:n {#1}
\__tag_struct_kid_mc_gput_right:nn
{ \g__tag_struct_stack_current_tl }
{#1}
\prop_gput:Nxx \g__tag_mc_parenttree_prop
{#1}
\end{verbatim}
Two commands to create artifacts, one without type, and one with. We define also a wrapper handler as luamode will need a different definition. TODO: perhaps later: more properties for artifacts

```
cs_new_protected:Npn \__tag_mc_bmc_artifact:n #1
  {
    \__tag_mc_bmc:n {Artifact}{/Type/#1}
  }

cs_new_protected:Npn \__tag_mc_handle_artifact:N #1
  {% #1 is a var containing the artifact type
   \int_gincr:N \c@g__tag_MCID_abs_int
   \tl_if_empty:NTF #1
   { \__tag_mc_bmc_artifact: }
   { \exp_args:NV\__tag_mc_bmc_artifact:n #1 }
  }
```

(End definition for \__tag_mc_bmc_artifact:, \__tag_mc_bmc_artifact:n, and \__tag_mc_handle_artifact:N.)

This allows to retrieve the active mc-tag. It is use by the get command.

```
cs_new:Nn \__tag_get_data_mc_tag: \g__tag_mc_key_tag_tl
```

(End definition for \__tag_get_data_mc_tag:)

These are the core public commands to open and close an mc. They don’t need to be in the same group or grouping level, but the code expect that they are issued linearly. The tag and the state is passed to the end command through a global var and a global boolean.

```
(base)\cs_new_protected:Npn \tag_mc_begin:n #1 \{ \__tag_whatsits: }
(base)\cs_new_protected:Nn \tag_mc_end:{ \__tag_whatsits: }
(generic|debug)
\cs_set_protected:Npn \tag_mc_begin:n #1 \%#1 keyval
  { \__tag_check_if_active_mc:T }
(generic)
\cs_set_protected:Npn \tag_mc_begin:n #1 \%#1 keyval
  { \__tag_check_if_active_mc:TF }
(debug)
\cs_set_protected:Npn \tag_mc_begin:n #1 \%#1 keyval
  { \__tag_debug_mc_begin_insert:n \{ #1 

71
\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 \g__tag_struct_tag_tl
\keys_set:nn { __tag / mc } {#1}
\bool_if:NTF \l__tag_mc_artifact_bool
\group_end:
\__tag_check_mc_if_nested:
\bool_gset_true:N \g__tag_in_mc_bool
\bool_if:NTF \l__tag_mc_artifact_bool
\group_end:

{ %handle artifact
\__tag_handle_artifact:N \l__tag_mc_artifact_type_tl
\exp_args:NV}
{ %handle mcid type
\__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
\__tag_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:n { \l__tag_mc_key_label_tl }
\bool_if:NF \l__tag_mc_key_stash_bool
{ \exp_args:NV \__tag_struct_get_tag_info:nNN
\l__tag_mc_key_tag_tl
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\__tag_check_parent_child:VVnnN
\l__tag_tmpa_tl \l__tag_tmpb_tl
{MC}()
\l__tag_parent_child_check_tl
\int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
{ \msg_warning:nnnn
{ tag }
{role-parent-child}
{ \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
{ MC(=real content) }
{not-allowed}. }
}
\__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
}
\group_end:
{ %handle artifact
\__tag_handle_artifact:N \l__tag_mc_artifact_type_tl
\exp_args:NV}
{ %handle mcid type
\__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
\__tag_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:n { \l__tag_mc_key_label_tl }
\bool_if:NF \l__tag_mc_key_stash_bool
{ \exp_args:NV \__tag_struct_get_tag_info:nNN
\l__tag_mc_key_tag_tl
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\__tag_check_parent_child:VVnnN
\l__tag_tmpa_tl \l__tag_tmpb_tl
{MC}()
\l__tag_parent_child_check_tl
\int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
{ \msg_warning:nnnn
{ tag }
{role-parent-child}
{ \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
{ MC(=real content) }
{not-allowed}. }
}
\__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
}
\group_end:
{ %handle artifact
\__tag_handle_artifact:N \l__tag_mc_artifact_type_tl
\exp_args:NV}
{ %handle mcid type
\__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
\__tag_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:n { \l__tag_mc_key_label_tl }
\bool_if:NF \l__tag_mc_key_stash_bool
{ \exp_args:NV \__tag_struct_get_tag_info:nNN
\l__tag_mc_key_tag_tl
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\__tag_check_parent_child:VVnnN
\l__tag_tmpa_tl \l__tag_tmpb_tl
{MC}()
\l__tag_parent_child_check_tl
\int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
{ \msg_warning:nnnn
{ tag }
{role-parent-child}
{ \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
{ MC(=real content) }
{not-allowed}. }
}
\__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
}
\group_end:
{ %handle artifact
\__tag_handle_artifact:N \l__tag_mc_artifact_type_tl
\exp_args:NV}
{ %handle mcid type
\__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
\__tag_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:n { \l__tag_mc_key_label_tl }
\bool_if:NF \l__tag_mc_key_stash_bool
{ \exp_args:NV \__tag_struct_get_tag_info:nNN
\l__tag_mc_key_tag_tl
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\__tag_check_parent_child:VVnnN
\l__tag_tmpa_tl \l__tag_tmpb_tl
{MC}()
\l__tag_parent_child_check_tl
\int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
{ \msg_warning:nnnn
{ tag }
{role-parent-child}
{ \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
{ MC(=real content) }
{not-allowed}. }
}
\__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
}
\group_end:

{ %handle artifact
\__tag_handle_artifact:N \l__tag_mc_artifact_type_tl
\exp_args:NV}
{ %handle mcid type
\__tag_check_mc_tag:N \l__tag_mc_key_tag_tl
\__tag_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:n { \l__tag_mc_key_label_tl }
\bool_if:NF \l__tag_mc_key_stash_bool
{ \exp_args:NV \__tag_struct_get_tag_info:nNN
\l__tag_mc_key_tag_tl
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\__tag_check_parent_child:VVnnN
\l__tag_tmpa_tl \l__tag_tmpb_tl
{MC}()
\l__tag_parent_child_check_tl
\int_compare:nNnT {\l__tag_parent_child_check_tl}<{0}
{ \msg_warning:nnnn
{ tag }
{role-parent-child}
{ \g__tag_struct_tag_tl/\g__tag_struct_tag_NS_tl }
{ MC(=real content) }
{not-allowed}. }
}
\__tag_mc_handle_stash:x { \int_use:N \c@g__tag_MCID_abs_int }
}
\group_end:

{ %debug
\__tag_debug_mc_begin_ignore:n { #1 }
\group_end:
{/debug}
1.4 Keys

Definitions are different in luamode. `tag` and `raw` are expanded as \lua_now:e in lua does it too and we assume that their values are safe.
(End definition for tag (mc-key) and others. These functions are documented on page 56.)
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\_@@\_mc\_type\_attr}: the value represent the type
\texttt{g\_@@\_mc\_cnt\_attr}: will hold the \texttt{\c@g\_@@\_MCID\_abs\_int} value

Handling attribute needs a different system to number the page wise mcid’s: a \texttt{\tagmcbegin \ldots \tagmcend} pair no longer surrounds exactly one mc chunk: it can be split at page breaks. We know the included mcid(s) only after the ship out. So for the \texttt{struct -> mc} mapping we need to record \texttt{struct -> mc\_cnt} (in \texttt{g\_@@\_mc\_parenttree\_prop} and/or a lua table and at shipout \texttt{mc\_cnt-> \{mcid, mcid, \ldots\}} 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 \texttt{ltx@@.mc[absnum]}. The fields of the table are:
tag : the type (a string)
raw : more properties (string)
label: a string.
artifact: the presence indicates an artifact, the value (string) is the type.
kids: a array of tables
\{1={kid=num2,page=pagenum1}, 2={kid=num2,page=pagenum2},\ldots\},
this describes the chunks the mc has been split to by the traversing code
parent: the number of the structure it is in. Needed to build the parent tree.

The main function which wanders through the shipout box to inject the literals. if the new callback is there, it is used.
\bool_if:NT\g__tag_active_space_bool
  {
    \lua_now:e
    {
      if-\luatexbase.callbacktypes.pre_shipout_filter-then-
      \luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)-
      \ltx.__tag.func.space_chars_shipout(TAGBOX)-return-true-
      \end-
      end-
    }
    \lua_now:e
    {
      \if-\luatexbase.callbacktypes.pre_shipout_filter-then-
      token.get_next()-
      \end-
      }\@secdotwo\@gobble
    {
      \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
      {
        \lua_now:e
        {
          \ltx.__tag.func.space_chars_shipout (tex.box["ShipoutBox"]) }
      }
    }
  }
\bool_if:NT\g__tag_active_mc_bool
  {
    \lua_now:e
    {
      if-\luatexbase.callbacktypes.pre_shipout_filter-then-
      \luatexbase.add_to_callback("pre_shipout_filter", function(TAGBOX)-
      \ltx.__tag.func.mark_shipout(TAGBOX)-return-true-
      \end-
      end-
    }
    \lua_now:e
    {
      \if-\luatexbase.callbacktypes.pre_shipout_filter-then-
      token.get_next()-
      \end-
      }\@secdotwo\@gobble
    {
      \hook_gput_code:nnn{shipout/before}{tagpdf/lua}
      {
        \lua_now:e
        {
          \ltx.__tag.func.mark_shipout (tex.box["ShipoutBox"]) }
      }
    }
  }

1.1 Commands

\__tag_add_missing_mcs_to_stream:Nn
This command is used in the output routine by the ptagging code. It should do nothing in luamode.
\cs_new_protected:Npn \__tag_add_missing_mcs_to_stream:Nn #1#2 {}

(End definition for \__tag_add_missing_mcs_to_stream:Nn.)

\__tag_mc_if_in_p:
\tag_mc_if_in:TF
\tag_mc_if_in:p F

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

\prg_new_conditional:Nnn \__tag_mc_if_in: {p,T,F,TF}
\int_compare:nNnTF
\prg_return_false:
\prg_return_true:

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

(End definition for \__tag_mc_if_in:TF and \tag_mc_if_in:TF. This function is documented on page 55.)

\__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
\cs_generate_variant:Nn \__tag_mc_lua_set_mc_type_attr:n { o }
\cs_new:Nn \__tag_mc_lua_unset_mc_type_attr:
\cs_new_protected:Nn \__tag_mc_lua_unset_mc_type_attr:n #1 #2 { }

%TODO ltx.__tag.func.get_num_from("#1") seems not to return a suitable number??
\tl_set:Nx \__tag_tmpa_tl { luatexbase.attributes.g__tag_mc_type_attr, \__tag_get_mc_abs_cnt: }
\tl_set:Nx \__tag_tmpb_tl { luatexbase.attributes.g__tag_mc_type_attr, \__tag_get_mc_abs_cnt: }
\tl_set:Nx \__tag_tmpc_tl { luatexbase.attributes.g__tag_mc_type_attr, \__tag_get_mc_abs_cnt: }
\tl_set:Nx \__tag_tmpd_tl { luatexbase.attributes.g__tag_mc_type_attr, \__tag_get_mc_abs_cnt: }

\lua_now:e
\lua_now:e
\lua_now:e
\lua_now:e
\lua_now:e
\lua_now:e
\lua_now:e
\lua_now:e
\lua_now:e
\lua_now:e

\text.setattribute
( "global",
luatexbase.attributes.g__tag_mc_type_attr,
\__tag_tmpa_tl
)
\text.setattribute
( "global",
luatexbase.attributes.g__tag_mc_type_attr,
\__tag_tmpb_tl
)
\text.setattribute
( "global",
luatexbase.attributes.g__tag_mc_type_attr,
\__tag_tmpc_tl
)
\text.setattribute
( "global",
luatexbase.attributes.g__tag_mc_type_attr,
\__tag_tmpd_tl
)

\text.print(\texattr{global}{luatexbase.attributes.g__tag_mc_type_attr})

\text.print(\texattr{global}{luatexbase.attributes.g__tag_mc_cnt_attr})

\text.setattribute(\text intox{\__tagtmpa_tl})
\texttt{\textbackslash tex.setattribute}\n\begin{verbatim}
(texturesetattribute
  ("global",
luatexbase.attributes.g__tag_mc_type_attr,
-2147483647
  )
)
\end{verbatim}
\texttt{\textbackslash lua_now:e}
\begin{verbatim}
{\texttt{\textbackslash tex.setattribute}
  ("global",
luatexbase.attributes.g__tag_mc_cnt_attr,
-2147483647
  )
}
\end{verbatim}

(End definition for \texttt{\_tag_mc_lua_set_mc_type_attr:n} and \texttt{\_tag_mc_lua_unset_mc_type_attr:}.)

\texttt{\_tag_mc_insert_mcid_kids:n}
\texttt{\_tag_mc_insert_mcid_single_kids:n}
These commands will in the finish code replace the dummy for a mc by the real mcid
kids we need a variant for the case that it is the only kid, to get the array right
\begin{verbatim}
\texttt{\cs_new:Nn \_tag_mc_insert_mcid_kids:n}
\begin{verbatim}
{\lua_now:e { ltx.__tag.func.mc_insert_kids (#1,0) }
}
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\texttt{\cs_new:Nn \_tag_mc_insert_mcid_single_kids:n}
\begin{verbatim}
{\lua_now:e {ltx.__tag.func.mc_insert_kids (#1,1) }
}
\end{verbatim}
\end{verbatim}

(End definition for \texttt{\_tag_mc_insert_mcid_kids:n} and \texttt{\_tag_mc_insert_mcid_single_kids:n}.)

\texttt{\_tag_mc_handle_stash:n}
\texttt{\_tag_mc_handle_stash:x}
This is the lua variant for the command to put an mcid absolute number in the current
structure.
\begin{verbatim}
\texttt{\cs_new:Nn \_tag_mc_handle_stash:n %1 mcidnum}
\begin{verbatim}
{\_tag_check_mc_used:n { #1 }
\seq_gput_right:cn % Don’t fill a lua table due to the command in the item,
% so use the kernel command
   { g__tag_struct_kids \g__tag_struct_stack_current_tl _seq }
   { \_tag_mc_insert_mcid_kids:n (#1) }
\lua_now:e { ltx.__tag.func.store_struct_mcabs
   (\g__tag_struct_stack_current_tl,#1)
}
\prop_gput:Nxx
\end{verbatim}
\end{verbatim}

78
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.

\tag_mc_begin:n
\cs_set_protected:Nn \tag_mc_begin:n
\group_begin: %__tag_check_if_active_mc:T
\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:n \l__tag_mc_key_label_tl}
% if not stashed record the absolute number
\bool_if:NF \l__tag_mc_key_stash_bool
{ \exp_args:NV \__tag_struct_get_tag_info:nNN
\g__tag_struct_stack_current_tl
\l__tag_tmpa_tl \l__tag_tmpb_tl
\__tag_check_parent_child:VVnnN \l__tag_tmpa_tl \l__tag_tmpb_tl
{MC}{}
\l__tag_parent_child_check_tl
\int_compare:nNnT \l__tag_parent_child_check_tl<0
{ \msg_warning:nnxxx
\cs_generate_variant:Nn \__tag_mc_handle_stash:n { x }
(End definition for \__tag_mc_handle_stash:n.)
{ tag }
{role-parent-child}
{/g_tag_struct_tag_tl/g_tag_struct_tag_WS_tl}
{MC (real content)}
{'not-allowed'.}

\_tag_mc_handle_stash:x \_tag_get_mc_abs_cnt: }
}

\group_end:
}

(End definition for \tag_mc_begin:n. This function is documented on page 55.)
\tag_mc_end: TODO: check how the use command must be guarded.
\cs_set_protected:Nn \tag_mc_end: {
\__tag_check_if_active_mc:T
{
%\__tag_check_mc_if_open:
\bool_gset_false:N \g__tag_in_mc_bool
\bool_set_false:N\l__tag_mc_artifact_bool
\__tag_mc_lua_unset_mc_type_attr:
\tl_set:Nn \l__tag_mc_key_tag_tl { }
\tl_gset:Nn \g__tag_mc_key_tag_tl { }
}
}

(End definition for \tag_mc_end:. This function is documented on page 55.)
\_tag_get_data_mc_tag: The command to retrieve the current mc tag. TODO: Perhaps this should use the attribute instead.
\cs_new:Npn \_tag_get_data_mc_tag: { \g__tag_mc_key_tag_tl }
(End definition for \_tag_get_data_mc_tag:)

1.2 Key definitions
TODO: check conversion, check if local/global setting is right.
\keys_define:nn {__tag / mc}
{
tag .code:n = %
{
\tl_set:Nx \l__tag_mc_key_tag_tl { #1 }
\tl_gset:Nx \g__tag_mc_key_tag_tl { #1 }
\lua_now:e
{
% ltx.__tag.func.store_mc_data(\_tag_get_mc_abs_cnt:,"tag","#1")
},
raw .code:n =
{
\tl_put_right:Nx \l__tag_mc_key_properties_tl { #1 }
}
\lua_now: e
  \ltx__tag.func.store_mc_data(\__tag_get_mc_abs_cnt:, "raw", "#1")
},

alt .code:n  = % Alt property
  {\str_set_convert:Noon
    \l__tag_tmpa_str
    { #1 }% default
    \{ utf16/hex \}
    \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
    \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }\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},

actualtext .code:n  = % Alt property
  {\str_set_convert:Noon
    \l__tag_tmpa_str
    { #1 }% default
    \{ utf16/hex \}
    \tl_put_right:Nn \l__tag_mc_key_properties_tl { /Alt~< }
    \tl_put_right:No \l__tag_mc_key_properties_tl { \l__tag_tmpa_str>~ }\lua_now: e
    {\ltx__tag.func.store_mc_data
    (\__tag_get_mc_abs_cnt:, "actualtext", "/ActualText~-\str_use:N \l__tag_tmpa_str")}
  }},

label .code:n =
  {\tl_set:Nn\l__tag_mc_key_label_tl { #1}\lua_now: e
    \ltx__tag.func.store_mc_data
    (\__tag_get_mc_abs_cnt:, "label", "#1")}
},

__artifact-store .code:n =
{ }
\lua_now:e
 {
 \ltx.__tag.func.store_mc_data 
 ( 
 \__tag_get_mc_abs_cnt:,"artifact","#1"
 )
 }
 }

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

{/luamode}

(End definition for tag (mc-key) and others. These functions are documented on page 56.)
Part VII

The \texttt{tagpdf-struct} module

Commands to create the structure

Part of the tagpdf package

1 Public Commands

\begin{verbatim}
\tag_struct_begin:n\{⟨key-values⟩\}
\tag_struct_end:
These commands start and end a new structure. They don’t start a group. They set all their values globally.

\tag_struct_use:n\{⟨label⟩\}
These commands insert a structure previously stashed away as kid into the currently active structure. A structure should be used only once, if the structure already has a parent a warning is issued.

\tag_struct_object_ref:n\{⟨struct number⟩\}
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 number⟩\}}. Be aware that it can only be used if the structure has already been created and that it doesn’t check if the object actually exists!

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

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

\tag_struct_parent_int:
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.
\end{verbatim}
2  Public keys

2.1  Keys for the structure commands

\texttt{tag_{(struct-key)}}  
This is required. The value of the key is normally one of the standard types listed in the main tagpdf documentation. It is possible to setup new tags/types. The value can also be of the form \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_{(struct-key)}}  
Normally a new structure inserts itself as a kid into the currently active structure. This key prohibits this. The structure is nevertheless from now on “the current active structure” and parent for following marked content and structures.

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

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

\texttt{alt_{(struct-key)}}  
This key inserts an /Alt value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once.

\texttt{actualtext_{(struct-key)}}  
This key inserts an /ActualText value in the dictionary of structure object. The value is handled as verbatim string and hex encoded. The value will be expanded first once.

\texttt{lang_{(struct-key)}}  
This key allows to set the language for a structure element. The value should be a bcp-identifier, e.g. de-De.
ref_{uni2423}(struct-key) This key allows to add references to other structure elements, it adds the /Ref array to the structure. The value should be a comma separated list of structure labels set with the label key. e.g. ref={label1,label2}.

E_{uni2423}(struct-key) This key sets the /E key, the expanded form of an abbreviation or an acronym (I couldn’t think of a better name, so I stuck to E).

AF_{uni2423}(struct-key) AF = ⟨object name⟩
AF-inline = ⟨text content⟩

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

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

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

AF can be used more than once, to associate more than one file. The inline keys can be used only once per structure. Additional calls are ignored.

attribute_{uni2423}(struct-key) This key takes as argument a comma list of attribute names (use braces to protect the commas from the external key-val parser) and allows to add one or more attribute dictionary entries in the structure object. As an example

\tagstructbegin{tag=TH,attribute= TH-row}
Attribute names and their content must be declared first in \tagpdfsetup.

attribute-class_{uni2423}(struct-key) This key takes as argument a comma list of attribute class names (use braces to protect the commas from the external key-val parser) and allows to add one or more attribute classes to the structure object.

Attribute class names and their content must be declared first in \tagpdfsetup.
2.2 Setup keys

\texttt{newattribute\_\{setup-key\}} \texttt{ newattribute = \{\langle name \rangle\}\{\langle Content \rangle\}}

This key can be used in the setup command \texttt{\textbackslash 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.

\texttt{\textbackslash tagpdfsetup}
\begin{verbatim}
{ newattribute =
 {TH-col}\{/O /Table /Scope /Column},
 newattribute =
 {TH-row}\{/O /Table /Scope /Row},
}
\end{verbatim}

\texttt{root-AF\_\{setup-key\}} \texttt{ root-AF = \langle object name \rangle}

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

3 Variables

\texttt{\textbackslash c@\_tag\_struct\_abs\_int}

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

\begin{verbatim}
\texttt{(base) newcounter \{ \_tag\_struct\_abs\_int \}}
\texttt{(base) int_gzero:N \c@\_tag\_struct\_abs\_int}
\end{verbatim}

(End definition for \texttt{c@\_tag\_struct\_abs\_int}.)

\texttt{\textbackslash \_tag\_struct\_objR\_seq}

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

\begin{verbatim}
\texttt{(\textbackslash header) \_tag\_seq\_new:N \_tag\_struct\_objR\_seq}
\end{verbatim}

(End definition for \texttt{\_tag\_struct\_objR\_seq}.)

\texttt{\_tag\_struct\_cont\_mc\_prop}

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

\begin{verbatim}
\texttt{\_tag\_prop\_new:N \_tag\_struct\_cont\_mc\_prop}
\end{verbatim}

(End definition for \texttt{\_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\}

(End definition for \g__tag_struct_stack_seq.)

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

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

(End definition for \g__tag_struct_tag_stack_seq.)

For the parent-child check, we need the tag and NS of every structure

\prop_new:N \g__tag_struct_tag_NS_prop

(End definition for \g__tag_struct_tag_NS_prop.)

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 \g__tag_struct_stack_parent_tmpa_tl

(End definition for \g__tag_struct_stack_current_tl and \g__tag_struct_stack_parent_tmpa_tl.)

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

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

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

All properties have at least the keys

**Type** StructTreeRoot or StructElem

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

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.

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

(End definition for \g__tag_struct_tag_tl and \g__tag_struct_tag_NS_tl.)

This will hold the label value.

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

(End definition for \l__tag_struct_key_label_tl.)

This will keep track of the stash status

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

(End definition for \l__tag_struct_elem_stash_bool.)
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:N \g__tag_struct_dest_num_prop

(End 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:N \g__tag_struct_ref_by_dest_prop

(End 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:Npn \__tag_struct_output_prop_aux:nn #1 #2 \%#1 num, #2 key

{ \prop_if_in:cnT { g__tag_struct_#1_prop } { #2 } { \c_space_tl/#2~ \prop_item:cn{ g__tag_struct_#1_prop } { #2 } } }

\cs_new_protected:Npn \__tag_new_output_prop_handler:n #1

{ \cs_new:cn { __tag_struct_output_prop_#1:n } \{ \__tag_struct_output_prop_aux:nn {#1}{##1} \} }

(End definition for \__tag_struct_output_prop_aux:nn and \__tag_new_output_prop_handler:n.)
4.1 Initialization of the StructTreeRoot

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

```latex
\tl_gset:Nn \_tag_struct_stack_current_tl {0}
```

```
\_tag_pdf_name:e:n
```

```latex
\cs_new:Npn \_tag_pdf_name:e:n #1{\pdf_name_from_unicode_e:n(#1)}
```

(End definition for \_tag_pdf_name:e:n.)

```latex
\__tag_prop_new:c { g__tag_struct_0_prop }
```

```latex
\__tag_new_output_prop_handler:n {0}
```

```latex
\__tag_seq_new:c { g__tag_struct_kids_0_seq }
```

```latex
\__tag_prop_gput:cnx
```

```latex
\prop_gput:Nnn \g__tag_struct_tag_NS_prop {0}{{StructTreeRoot}{pdf}}
```

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

```latex
\__tag_prop_gput:cnx
```

(End definition for g__tag_struct_0_prop and g__tag_struct_kids_0_seq.)

4.2 Filling in the tag info

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

```latex
\pdf_version_compare:NnTF < {2.0} {
\cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
```

```
{ g__tag_struct_0_prop }
```

```
{ \pdf_name_from_unicode_e:n {StructTreeRoot} }
```

```latex
\prop_gput:Nnn \g__tag_struct_tag_NS_prop {0}{{StructTreeRoot}{pdf}}
```

(End definition for g__tag_struct_0_prop and g__tag_struct_kids_0_seq.)
\cs_new_protected:Npn \__tag_struct_set_tag_info:nnn #1 #2 #3
\{ \__tag_prop_gput:cnx
\{ g__tag_struct_#1_prop \}
\{ S \}
\{ \pdf_name_from_unicode_e:n \{#2\} \%
\prop_get:NnNT \g__tag_role_NS_prop \{#3\} \l__tag_get_tmpc_tl
\}
\__tag_prop_gput:cnx
\{ g__tag_struct_#1_prop \}
\{ NS \}
\{ \l__tag_get_tmpc_tl \%
\}
\prop_gput:Nnn \g__tag_struct_tag_NS_prop \{#1\}\{#2\}\{#3\}
\}
\cs_generate_variant:Nn \__tag_struct_set_tag_info:nnn {eVV}

\cc We also need a way to get the tag info back from parent structures.
\cs_new_protected:Npn \__tag_struct_get_tag_info:nNN #1 #2 #3
\{ \prop_get:NnNTF \g__tag_struct_tag_NS_prop \{#1\}\l__tag_get_tmpc_tl
\{ \tl_set:Nx \l__tag_get_tmpc_tl{\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
\tl_set:Nx \l__tag_get_tmpc_tl{\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
\}
\{ \tl_clear:N\l__tag_get_tmpc_tl
\tl_clear:N\l__tag_get_tmpc_tl
\}
\cs_generate_variant:Nn \__tag_struct_get_tag_info:nNN {eNN}

(End definition for \__tag_struct_set_tag_info:nnn.)

\cc We also need a way to get the tag info back from parent structures.
\cs_new_protected:Npn \__tag_struct_set_tag_info:nNN \{#1\} \l__tag_get_tmpc_tl
\{ \tl_set:Nx \l__tag_get_tmpc_tl{\exp_last_unbraced:NV\use_i:nn \l__tag_get_tmpc_tl}
\tl_set:Nx \l__tag_get_tmpc_tl{\exp_last_unbraced:NV\use_ii:nn \l__tag_get_tmpc_tl}
\}
\{ \tl_clear:N\l__tag_get_tmpc_tl
\tl_clear:N\l__tag_get_tmpc_tl
\}
\cs_generate_variant:Nn \__tag_struct_set_tag_info:nNN \{eNN}

(End definition for \__tag_struct_set_tag_info:nnn.)

\section{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!
\cs_new:Npn \__tag_struct_mcid_dict:n #1 %#1 MCID absnum
{
<< /Type \c_space_tl /MCR \c_space_tl
/Pg \c_space_tl
\pdf_pageobject_ref:n { \__tag_ref_value:enn{mcid-#1}{tagabspage}{1} }
/MCID \c_space_tl \__tag_ref_value:enn{mcid-#1}{tagmcid}{1}
>>
}
\cs_new_protected:Npn \__tag_struct_kid_mc_gput_right:nn #1 #2 %#1 structure num, #2 MCID absnum%
{
\__tag_seq_gput_right:cx
{ g__tag_struct_kids_#1_seq }
\__tag_struct_mcid_dict:n {#2}
\__tag_seq_gput_right:cn
{ g__tag_struct_kids_#1_seq }
\prop_item:Nn \g__tag_struct_cont_mc_prop {#2}
}
\cs_generate_variant:Nn \__tag_struct_kid_mc_gput_right:nn {nx}
(End definition for \__tag_struct_kid_mc_gput_right:nn.)

\__tag_struct_kid_struct_gput_right:nn
\__tag_struct_kid_struct_gput_right:xx
This commands adds a structure as kid. We only need to record the object reference in
the sequence.
\cs_new_protected:Npn \__tag_struct_kid_struct_gput_right:nn #1 #2 %#1 num of parent struct, #2 kid struct
{
\__tag_seq_gput_right:cx
{ g__tag_struct_kids_#1_seq }
\__tag_struct_struct_gput_right:xx
{ pdf_object_ref:n { __tag/struct/#2 } }
}
\cs_generate_variant:Nn \__tag_struct_kid_struct_gput_right:nn {xx}
(End definition for \__tag_struct_kid_struct_gput_right:nn.)

\__tag_struct_kid_OBJR_gput_right:nnn
\__tag_struct_kid_OBJR_gput_right:xxx
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
\cs_new_protected:Npn \__tag_struct_kid_OBJR_gput_right:nnn #1 #2 #3 %#1 num of parent struct, #2
\%#2 obj reference #3 page object reference
{
}\pdf_object_unnamed_write:nn
{ dict }
{ /Type/OBJR/Obj~#2/Pg~#3

92
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
cids to mark this case.

\cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1
\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:Nx \l__tag_tmpa_seq { ##1 }
\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_gset_eq:cN { g__tag_struct_kids_#1_seq } \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 }
\g__tag结构_kid_OBJR_gput_right:nnn { xxx }
\cs_generate_variant:Nn \__tag_struct_kid_OBJR_gput_right:nnn { xxx }
(End definition for \__tag_struct_kid_OBJR_gput_right:nnn.)
\__tag_struct_exchange_kid_command:N
\__tag_struct_exchange_kid_command:c
This command adds the kid info to the K entry. In lua mode the content contains
commands which are expanded later. The argument is the structure number.
\cs_new_protected:Npn \__tag_struct_fill_kid_key:n #1
\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:Nx \l__tag_tmpa_seq { ##1 }
\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_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 }
\g__tag结构_kid_OBJR_gput_right:nnn { xxx }
\cs_generate_variant:Nn \__tag_struct_kid_OBJR_gput_right:nnn { xxx }
(End definition for \__tag_struct_kid_OBJR_gput_right:nnn.)
\__tag_struct_exchange_kid_command:N
\__tag_struct_exchange_kid_command:c
{ 0 } %no kids, do nothing
{ 1 } % 1 kid, insert
{ % in this case we need a special command in
% luamode to get the array right. See issue #13
\bool_if:NT\g__tag_mode_lua_bool
{ \__tag_struct_exchange_kid_command:c
{g__tag_struct_kids_#1_seq} }
\__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
{ seq_item:cn
    { g__tag_struct_kids_#1_seq
    {1} }
} %
}
{ %many kids, use an array
\__tag_prop_gput:cnx { g__tag_struct_#1_prop } {K}
{ [ seq_use:cn
    { g__tag_struct_kids_#1_seq
    } { c_space_tl }
] }
} %
}
(End definition for \__tag_struct_fill_kid_key:n.)

4.4 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
\c__tag_struct_
\int_compare:nNnTF{#1}={0}{StructTreeRoot}{StructElem}
_entries_seq
Some keys needs the option to format the key, e.g. add brackets for an array.

```latex
\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 }
\c_space_tl/##1~
```

(End definition for \_tag_struct_get_dict_content:nN.)

\_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:

```latex
\cs_new:Nn\_tag STRUCT_FORMAT_REF:n{[#1]}
\cs_generate_variant:Nn\_tag STRUCT_FORMAT_REF:n{e}
```

(End definition for \_tag STRUCT_FORMAT_REF:n.)

\_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.

```latex
\cs_new_protected:Npn \_tag STRUCT_WRITE_OBJ:n #1 % #1 is the struct num
  \pdf_object_if_exist:nTF {__tag/struct/#1} 
  \_tag STRUCT_FILL_KID_KEY:n { #1 }
  \_tag STRUCT_GET_DICT_CONTENT:nN { #1 } \l__tag_tmpa_tl
\exp_args:Nx\pdf_object_write:nnx
\_tag STRUCT_WRITE:nnx
  \tag/struct/#1
  \l__tag_tmpa_tl
\msg_error:nnn { tag } { struct-no-objnum } { #1}
```

(End definition for \_tag STRUCT_WRITE_OBJ: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 }

(1) \pdfannot_dict_put:nnx
\link/URI
\StructParent
\int_use:N\c@g_@@_parenttree_obj_int

<start link> link text <stop link>

(2+3) \@@_struct_insert_annot:nn {obj ref}{parent num}
\tag mc_end:
\tag struct_end:

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

\__tag_get_data_struct_tag: this command allows \tag_get:n to get the current structure tag with the keyword \texttt{struct_tag}. We will need to handle nesting
\cs_new:Npn \__tag_get_data_struct_tag: {
\exp_args:Ne \tl_tail:n {
\prop_item:cn {g__tag_struct_\g__tag_struct_stack_current_tl_prop}{}{S}
\}
\}
\__tag_get_data_struct_num: this command allows \tag_get:n to get the current structure number with the keyword \texttt{struct_num}. We will need to handle nesting
\cs_new:Npn \__tag_get_data_struct_num: {
\g__tag_struct_stack_current_tl
\}
\keys_define:nn { __tag / struct }
\key{label}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{stash}{\bool_set:N = \l__tag_struct_elem_stash_bool,}
\key{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}{}}}
\key{title}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{title-o}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{alt}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{actualtext}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{lang}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{ref}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{E}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{E}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{lang}{\tl_set:N = \l__tag_struct_key_label_tl,}
\key{ref}{\tl_set:N = \l__tag_struct_key_label_tl,}

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.

\begin{verbatim}
\end{verbatim}
\tl_set:Nx \l__tag_struct_stack_parent_tmpa_tl \{ \int_eval:n \{#1\} \}
{
\msg_warning:nnxx { tag } { struct-unknown }
{ \int_eval:n \{#1\} }
{ parent-key-ignored }
}
parent .default:n = {-1},
tag .code:n = % S property
{ \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Ne\g__tag_role_tags_NS_prop{#1}}
\tl_get:Nx \g__tag_struct_tag_tl \{ \seq_item:Nn\l__tag_tmpa_seq {1} \}
\tl_get:Nx \g__tag_struct_tag_NS_tl{ \seq_item:Nn\l__tag_tmpa_seq {2} }
\__tag_check_structure_tag:N \g__tag_struct_tag_tl
},
title .code:n = % T property
{ \str_set_convert:Nnnn
 \l__tag_tmpa_str
 { #1 }
 { default }
 { utf16/hex }
 \__tag_prop_gput:cnx
 { g__tag_struct\_int_eval:n \{c@g__tag_struct_abs_int\}_prop }
 { T }
 { <\l__tag_tmpa_str> }
},
title-o .code:n = % T property
{ \str_set_convert:Nnnn
 \l__tag_tmpa_str
 { #1 }
 { default }
 { utf16/hex }
 \__tag_prop_gput:cnx
 { g__tag_struct\_int_eval:n \{c@g__tag_struct_abs_int\}_prop }
 { T }
 { <\l__tag_tmpa_str> }
},
alt .code:n = % Alt property
{ \str_set_convert:Nnnn
 \l__tag_tmpa_str
 { #1 }
 { default }
 { utf16/hex }
 \__tag_prop_gput:cnx
 { g__tag_struct\_int_eval:n \{c@g__tag_struct_abs_int\}_prop }
 { Alt }
 { <\l__tag_tmpa_str> }
},
alttext .meta:n = {alt=#1},
actualtext .code:n = % ActualText property
{
\str_set_convert:Noon
\l__tag_tmpa_str
  \{ #1 \}
  \{ default \}
  \{ utf16/hex \}
\__tag_prop_gput:cnx
  \{ g__tag_struct\_int_eval:n \{c@g__tag_struct_abs_int\}_prop \}
  \{ ActualText \}
  \{ <\l__tag_tmpa_str>\},
\lang .code:n = % Lang property
{ \__tag_prop_gput:cnx
  \{ g__tag_struct\_int_eval:n \{c@g__tag_struct_abs_int\}_prop \}
  \{ Lang \}
  \{ (#1) \},
},

Ref 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:Nx \l__tag_tmpa_tl
  \{~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} \}
  \} \__tag_struct_gput_data_ref:ee \{ \int_eval: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_prop_gput:cnx
  \{ g__tag_struct\_int_eval:n \{c@g__tag_struct_abs_int\}_prop \}
  \{ E \}
  \{ <\l__tag_tmpa_str> \}
},

(End definition for label (struct-key) and others. These functions are documented on page 84.)

AF_{\text{struct-key}}
AFinline_{\text{struct-key}}
AFinline-o_{\text{struct-key}}

keys for the AF keys (associated files). They use commands from l3pdfx! The stream
variants use txt as extension to get the mimetype. TODO: check if this should be
configurable. For math we will perhaps need another extension. AF is an array and can
be used more than once, so we store it in a tl. which is expanded. AFinline can be use
only once (more quite probably doesn’t make sense).
\cs_new_protected:Npn \__tag_structure_add_AF:nn #1 #2 % #1 struct num #2 object name
{ \tl_if_exist:cTF
  \g__tag_struct_#1_AF_tl

\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:nTF {#1}
{ \__tag_struct_add_AF:en \int_eval:n {\c@g__tag_struct_abs_int} }{#1}
\__tag_prop_gput:cnx
{ g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
{ AF }
{ [\tl_use:c
{ g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
]}
}
}
,AFinline .code:n =
{ \group_begin:
\pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
{ \pdffile_embed_stream:nxx
{\#1}
{tag-AFfile\int_use:N\c@g__tag_struct_abs_int.txt}
{\_tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
\__tag_struct_add_AF:ee
{ \int_eval:n {\c@g__tag_struct_abs_int} }
{ \_tag/fileobj\int_use:N\c@g__tag_struct_abs_int }
\__tag_prop_gput:cnx
{ g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
{ AF }
{ [\tl_use:c

100
{ g__tag_struct__int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
}
\group_end:
\group_end:

,AFinline-o .code:n =
{
\group_begin:
\pdf_object_if_exist:eF {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
{
pdffile_embed_stream:oxx
{#1}
{__tag/fileobj\int_use:N\c@g__tag_struct_abs_int}
\__tag_struct_add_AF:ee
{ \int_eval:n {\c@g__tag_struct_abs_int} }
{ __tag/fileobj\int_use:N\c@g__tag_struct_abs_int }
\__tag_prop_gput:cnx
{ g__tag_struct__int_use:N\c@g__tag_struct_abs_int__prop }
{ AF }
{ }
{ \tl_use:c
{ g__tag_struct__int_eval:n {\c@g__tag_struct_abs_int}_AF_tl }
}
}
\group_end:
\group_end:

(End definition for AF (struct-key), AFinline (struct-key), and AFinline-o (struct-key). These functions are documented on page 85.)

_root-AF_(setup-key) The root structure can take AF keys too, so we provide a key for it. This key is used with \tagpdfsetup, not in a structure!

\keys_define:nn { __tag / setup }
{ root-AF .code:n =
{ \pdf_object_if_exist:nTF {#1}
{ \__tag_struct_add_AF:en { 0 }{#1}
\__tag_prop_gput:cnx
{ g__tag_struct_0_prop }
{ AF }
{ }
{ \tl_use:c
{ g__tag_struct_0_AF_tl }
}
}
6 User commands

\begin{verbatim}
\tag_struct_begin:n
\tag_struct_end:
\tag_struct_begin:n #1 \{ \int_gincr:N \c@g__tag_struct_abs_int \}
\tag_struct_end:
\tag_struct_begin:n #1 \#1 \#1 \key-val
\tag_struct_begin:n #1 \#1 \#1 \key-val
\__tag_check_if_active_struct:T
\__tag_check_if_active_struct:TF
\group_begin:
\int_gincr:N \c@g__tag_struct_abs_int
\__tag_prop_new:c { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
\__tag_new_output_prop_handler:n { \int_eval:n { \c@g__tag_struct_abs_int } }
\__tag_seq_new:c { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
\exp_args:Ne
\pdf_object_new:n
{ __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
\__tag_prop_gput:cno
{ g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
{ Type }
{ /StructElem }
\tl_set:Nn \l__tag_struct_stack_parent_tmpa_tl {-1}
\keys_set:nn { __tag / struct } { #1 }
\__tag_struct_set_tag_info:eVW
{ \int_eval:n { \c@g__tag_struct_abs_int } }
\g__tag_struct_tag_t1
\g__tag_struct_tag_NS_t1
\__tag_check_structure_has_tag:n { \int_eval:n { \c@g__tag_struct_abs_int } }
\tl_if_empty: NF
\l__tag_struct_key_label_t1
{ }
\__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_t1}{struct}
\}
\\end{verbatim}

The structure number of the parent is either taken from the stack or has been set with the parent key.

\begin{verbatim}
\int_compare:nNnT \{ \l__tag_struct_stack_parent_tmpa_tl \} = \{ -1 \}
\seq_set:NF
\g__tag_struct_stack_seq
\l__tag_struct_stack_parent_tmpa_tl
\end{verbatim}
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\_tag\_info:\enN
\l\_tag\_get\_parent\_tmpa\_tl
\l\_tag\_get\_parent\_tmpb\_tl
\_tag\_check\_parent\_child:VVVVNN
\l\_tag\_get\_parent\_tmpa\_tl
\l\_tag\_get\_parent\_tmpb\_tl
\g\_tag\_struct\_tag\_tl
\g\_tag\_struct\_tag\_NS\_tl
\l\_tag\_parent\_child\_check\_tl
\int\_compare:nnnT {\l\_tag\_parent\_child\_check\_tl}<0
{
\msg\_warning:nnnxx
{ tag }
{role-parent-child}
{ \l\_tag\_get\_parent\_tmpa\_tl/\l\_tag\_get\_parent\_tmpb\_tl }
{ \g\_tag\_struct\_tag\_tl/\g\_tag\_struct\_tag\_NS\_tl }
{ not-allowed-(struct-\g\_tag\_struct\_stack\_current\_tl) }
\cs\_set\_eq:nnN \l\_tag\_role\_remap\_tag\_tl \g\_tag\_struct\_tag\_tl
\cs\_set\_eq:nnN \l\_tag\_role\_remap\_NS\_tl \g\_tag\_struct\_tag\_NS\_tl
\_tag\_role\_remap:
\cs\_get\_eq:nnN \g\_tag\_struct\_tag\_tl \l\_tag\_role\_remap\_tag\_tl
\cs\_get\_eq:nnN \g\_tag\_struct\_tag\_NS\_tl \l\_tag\_role\_remap\_NS\_tl
\_tag\_struct\_set\_tag\_info:eeV
{ \int\_eval:n {\c\_tag\_struct\_abs\_int} }
\g\_tag\_struct\_tag\_tl
\g\_tag\_struct\_tag\_NS\_tl
}

Set the Parent.

\_tag\_prop\_gput:cnx
{ \g\_tag\_struct\_int\_eval:n {\c\_tag\_struct\_abs\_int}\_prop }
\P
{
\pdf\_object\_ref:e { \_tag/struct/\l\_tag\_struct\_stack\_parent\_tmpa\_tl }
}
%record this structure as kid:
%\tl\_show:N \g\_tag\_struct\_stack\_current\_tl
%\tl\_show:N \l\_tag\_struct\_stack\_parent\_tmpa\_tl
\_tag\_struct\_kid\_struct\_gput\_right:xx
\tagstruct\ Use This command allows to use a stashed structure in another place. TODO: decide how it should be guarded. Probably by the struct-check.

\tagstructure This command allows to use a stashed structure in another place. TODO: decide how it should be guarded. Probably by the struct-check.
\prop_if_exist:cTF
\begin{verbatim}
g__tag_struct__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{unknown}_prop
\end{verbatim}
\begin{verbatim}
\__tag_check_struct_used:n {#1}
\__tag_struct_kid_struct_gput_right:xx
\end{verbatim}
\begin{verbatim}
g__tag_struct__tag_ref_value:enn{tagpdfstruct-#1}{tagstruct}{0}_prop
\end{verbatim}
\begin{verbatim}
\__tag_check_parent_child:VVVVN
\end{verbatim}
\begin{verbatim}
\int_compare:nNnT {\l__tag_parent_child_check_tl}<0
\begin{verbatim}
\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:
\end{verbatim}
\begin{verbatim}
g__tag_struct_tag_tl
g__tag_struct_tag_NS_tl
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\msg_warning:nnn{ tag }{struct-label-unknown}{#1}
\end{verbatim}
\end{verbatim}

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:

\begin{verbatim}
\__tag_struct_get_tag_info:ENN
\end{verbatim}
\begin{verbatim}
\l__tag_tmpa_tl
\l__tag_tmpb_tl
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\int_compare:nNnT {\l__tag_parent_child_check_tl}<0
\begin{verbatim}
\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:
\end{verbatim}
\begin{verbatim}
g__tag_struct_tag_tl
g__tag_struct_tag_NS_tl
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\msg_warning:nnn{ tag }{struct-label-unknown}{#1}
\end{verbatim}
\end{verbatim}

(End definition for \tag_struct_use:n. This function is documented on page 83.)

\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.
\texttt{\_tag\_struct\_gput:nnn} This is a command that allows to update the data of a structure. 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}.

\texttt{\_tag\_struct\_insert\_annot:nnn, \_tag\_struct\_insert\_annot:xx \_tag\_struct\_parent\_int:} This are the user command to insert annotations. They must be used together to get the numbers right. They use a counter to the \texttt{StructParent} and \texttt{\_tag\_struct\_insert\_annot:nnn} increases the counter given back by \texttt{\_tag\_struct\_parent\_int:}. It must be used together with \texttt{\_tag\_struct\_parent\_int:} to insert an annotation. TODO: decide how it should be guarded if tagging is deactivated.
7 Attributes and attribute classes

\section{Variables}
\begin{itemize}
\item \texttt{\g__tag_attr_entries_prop} will store attribute names and their dictionary content.
\item \texttt{\g__tag_attr_class_used_seq} will hold the attributes which have been used as class name.
\item \texttt{\l__tag_attr_value_tl} is used to build the attribute array or key. Everytime an attribute is used for the first time, and object is created with its content, the name-object reference relation is stored in \texttt{\g__tag_attr_objref_prop}
\end{itemize}

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

\begin{verbatim}
\ Pendfsetup
{   \newattribute =
    {TH-col}{/O /Table /Scope /Column},
   \newattribute =
    {TH-row}{/O /Table /Scope /Row},
}
\end{verbatim}
newattribute .code:n =
\__tag_attr_new_entry:nn #1
}

(End definition for \__tag_attr_new_entry:nn and newattribute (setup-key). This function is documented on page 86.)

attribute-class\{struct-key\} attribute-class has to store the used attribute names so that they can be added to the ClassMap later.
\keys_define:nn { __tag / struct }
\attribute-class .code:n =
\clist_set:No \l__tag_tmpa_clist { #1 }
\seq_set_from_clist:NN \l__tag_tmpb_seq \l__tag_tmpa_clist
we convert the names into pdf names with slash
\seq_map_inline:Nn \l__tag_tmpa_seq
\pdf_name_from_unicode_e:n {##1}
\seq_gput_left:Nn \g__tag_attr_class_used_seq {##1}
\tl_set:Nx \l__tag_tmpa_tl
\int_compare:nT \seq_count:N \l__tag_tmpa_seq > 1 { [ }
\seq_map_inline:Nn \l__tag_tmpa_seq { \c_space_tl }
\int_compare:nT \seq_count:N \l__tag_tmpa_seq > 1 { ]}
%
\__tag_prop_gput:cnx \g__tag_struct_abs_int_prop \l__tag_tmpa_tl
(End definition for attribute-class (struct-key). This function is documented on page 85.)

attribute\{struct-key\}
\keys_define:nn { __tag / struct }
\attribute .code:n = % A property (attribute, value currently a dictionary)
we convert the names into pdf names with slash

\seq_set_map_x:NNn \l__tag_tmpa_seq \l__tag_tmpb_seq
{ \pdf_name_from_unicode_e:n {##1}\}
\tl_set:Nx \l__tag_attr_value_tl
{ \int_compare:nN { \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} { %%1 }
    \prop_if_in:NnF \g__tag_attr_objref_prop {##1}{
      \pdf_object_unnamed_write:nx
dict
      \prop_item:Nn \g__tag_attr_entries_prop {##1}
      \prop_gput:Nnx \g__tag_attr_objref_prop {##1} \pdf_object_ref_last:
    }%
    \tl_put_right:Nx \l__tag_attr_value_tl
    \c_space_tl
    \prop_item:Nn \g__tag_attr_objref_prop {##1}%
  }
  % \tl_show:N \l__tag_attr_value_tl
}{%
  \int_compare:nN { \seq_count:N \l__tag_tmpa_seq > 1 }{%}\%
}{%
  \tl_show:N \l__tag_attr_value_tl
  \tag_prop_gput:cnx\g__tag_struct_int\prop
  { A }\c_apx_one
  \l__tag_attr_value_tl
},\paste_angle
{/package}

(End definition for attribute (struct-key). This function is documented on page 85.)
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.

\begin{verbatim}
\fontencoding{TU}\fontfamily{lmr}\fontseries{m}\fontshape{n}\fontsize{10pt}{10pt}\selectfont
\let\lua_now:e\ltxdefineline
\lua_now:e { tagpdf=require('tagpdf.lua') }
\end{verbatim}

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 \texttt{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 \texttt{ltx.@@.tables[string]} Old code, I’m
not quite sure if this was a good idea. Now I have mix of table in \texttt{ltx.@@.tables} and
\texttt{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!

\begin{verbatim}
\cs_set_protected:Npn \__tag_prop_new:N #1
  \{ \prop_new:N #1 \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} } \}
\cs_set_protected:Npn \__tag_seq_new:N #1
  \{ \seq_new:N #1 \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1 = {} } \}
\cs_set_protected:Npn \__tag_prop_gput:Nnn #1 #2 #3
  \{ \prop_gput:Nnn #1 { #2 } { #3 } \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1["#2"] = "#3" } \}
\cs_set_protected:Npn \__tag_seq_item:cn #1 #2 \__tag_seq_show:N #1
  \{ \seq_show:N #1 \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1["#2"] = {} } \}
\cs_set_protected:Npn \__tag_seq_gput_right:Nn #1 #2
  \{ \seq_gput_right:Nn #1 { #2 } \lua_now:e { ltx.__tag.tables.\cs_to_str:N#1["#2"] = {} } \}
\end{verbatim}
\cs_set_protected:Npn __tag_seq_gput_right:Nn #1 #2
\seq_gput_right:Nn #1 { #2 }
\lua_now:e { table.insert(ltx.__tag.tables.\cs_to_str:N#1, "#2") }

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

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

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

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

(End definition for __tag_prop_new:N and others.)

(\luatex)

The module declaration

(*lua)
-- tagpdf.lua
-- Ulrike Fischer

local ProvidesLuaModule = {
  name = "tagpdf",
  version = "0.98", --TAGVERSION
  date = "2022-12-22", --TAGDATE
  description = "tagpdf lua code",
  license = "The LATEX Project Public License 1.3c"
}

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

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

Some comments about the lua structure.

the main table is named ltx.__tag. It contains the functions and also the data collected during the compilation.
ltx.__tag.mc will contain mc connected data.
ltx.__tag.struct will contain structure related data.
ltx.__tag.page will contain page data
ltx.__tag.tables contains also data from mc and struct (from older code). This needs cleaning
There are certainly doublutes, but I don’t dare yet ...
ltx.__tag.func will contain (public) functions.
ltx.__tag.trace will contain tracing/logging functions.
local functions start with __
functions meant for users will be in ltx.tag

functions
ltx.__tag.func.get_num_from (tag): takes a tag (string) and returns the id number
ltx.__tag.func.output_num_from (tag): takes a tag (string) and prints (to tex) the id number
ltx.__tag.func.get_tag_from (num): takes a num and returns the tag
ltx.__tag.func.output_tag_from (num): takes a num and prints (to tex) the tag
ltx.__tag.func.store_mc_data (num,key,data): stores key=data in ltx.__tag.mc[num]
ltx.__tag.func.store_mc_label (label,num): stores label=num in ltx.__tag.mc.labels
ltx.__tag.func.store_mc_kid (mcnum,kid,page): stores the mc-kids of mcnum on page page
ltx.__tag.func.store_mc_in_page(mcnum,mcpagecnt,page): stores in the page table the number of mcnum on this page
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,mcopen,name,mctypeprev) : the main ltx.__tag.func.mark_page_elements(box,mcpagecnt,mcopen,name,mctypeprev) : the main function
ltx.__tag.func.store_struct_data (num): stores the struct data of structnum
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 positions
ltx.__tag.func.show_mc_data (num,loglevel): shows ltx.__tag.mc[num] is 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.showprop: shows a prop
ltx.__tag.trace.log
ltx.__tag.trace.showspaces : boolean

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

```
local mctypeattributeid = luatexbase.new_attribute("g\_tag\_mc\_type\_attr")
local mccntattributeid = luatexbase.new_attribute("g\_tag\_mc\_cnt\_attr")
local iwspacettributeid = luatexbase.new_attribute("g\_tag\_interwordspace\_attr")
local iwfontattributeid = luatexbase.new_attribute("g\_tag\_interwordfont\_attr")
```

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

```
local tagunmarkedbool = token.create("g\_tag\_tagunmarked\_bool")
local truebool = token.create("c\_true\_bool")
```

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

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

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

```
ltx = ltx or { }
ltx.__tag = ltx.__tag or { } -- mc data
ltx.__tag.struct = ltx.__tag.struct or { } -- struct data
ltx.__tag.tables = ltx.__tag.tables or { } -- tables created with new prop and new seq.
-- wasn’t a so great idea ...
-- g__tag\_role\_tags\_seq used by tag\<\> is in this table
-- used for lua tables too now!
ltx.__tag.page = ltx.__tag.page or { } -- page data, currently only i\->{0->mcnum,1->mcnum}
ltx.__tag.trace = ltx.__tag.trace or { } -- show commands
```

113
2 Logging functions

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.

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

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

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

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

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

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

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

```
ltx.__tag.trace.show_prop (prop)
```

114
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 definition for __tag_pairs_prop and ltx.__tag.trace.show_prop.)

ltx.__tag.trace.show_mc_data
This shows some data for a mc given by num. If something is shown depends on the log level. The function is used by the following function and then in 
>ShowTagging
function ltx.__tag.trace.show_mc_data (num,loglevel)
    if ltx.__tag and ltx.__tag.mc and ltx.__tag.mc[num] then
        for k,v in pairs(ltx.__tag.mc[num]) do
            __tag_log ("mc"..num..": ".tostring(k).."=>"..tostring(v),loglevel)
        end
        if ltx.__tag.mc[num]["kids"] then
            __tag_log ("mc" .. num .. " has ".#ltx.__tag.mc[num]["kids"] .. " kids",loglevel)
            for k,v in ipairs(ltx.__tag.mc[num]["kids"]) do
                __tag_log ("mc ".num .. " kid ".k .. "=>" .. v.kid .. " on page " ..v.page,loglevel)
            end
        end
        else
            __tag_log ("mc" ..num.." not found",loglevel)
        end
    end
end

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

ltx.__tag.trace.show_all_mc_data
This shows data for the mc's between min and max (numbers). It is used by the ShowTagging function.
function ltx.__tag.trace.show_all_mc_data (min,max,loglevel)
    for i = min, max do
        ltx.__tag.trace.show_mc_data (i,loglevel)
    end
texio.write_nl"
end

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

ltx.__tag.trace.show_struct_data
This function shows some struct data. Unused but kept for debugging.
function ltx.__tag.trace.show_struct_data (num)
    if ltx.__tag and ltx.__tag.struct and ltx.__tag.struct[num] then
        for k,v in ipairs(ltx.__tag.struct[num]) do
            __tag_log ("struct ".num..": ".tostring(k).."=>"..tostring(v),1)
        end
    else
        __tag_log ("struct ".num.." not found ",1)
    end
end

(End definition for ltx.__tag.trace.show_struct_data.)
3 Helper functions

3.1 Retrieve data functions

__tag_get_mc_cnt_type_tag

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

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

(End definition for __tag_get_mc_cnt_type_tag.)

__tag_get_mathsubtype

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

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

(End definition for __tag_get_mathsubtype.)

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.

```lua
ltx.__tag.tables.role_tag_attribute = {}
```

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

ltx.__tag.func.alloctag

```lua
local __tag_alloctag =
    function (tag)
        if not ltx.__tag.tables.role_tag_attribute[tag] then
            table.insert(ltx.__tag.tables.role_attribute_tag,tag)
            ltx.__tag.tables.role_tag_attribute[tag]=#ltx.__tag.tables.role_attribute_tag
            __tag_log ("Add ".tag.." ".ltx.__tag.tables.role_tag_attribute[tag],3)
        end
    end
end
```

ltx.__tag.func.alloctag = __tag_alloctag

(End definition for ltx.__tag.func.alloctag.)

ltx.__tag.func.get_num_from

ltx.__tag.func.get_num_from

ltx.__tag.func.output_num_from

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

```lua
local __tag_get_num_from,
    function (tag)
        if ltx.__tag.tables.role_tag_attribute[tag] then
```

116
a = ltx.__tag.tables.role_tag_attribute[tag]
else
  a = -1
end
return a
end

ltx.__tag.func.get_num_from = __tag_get_num_from

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

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

__tag_get_tag_from

ltx.__tag.func.get_tag_from = __tag_get_tag_from

function (num)
  if ltx.__tag.tables.role_attribute_tag[num] then
    a = ltx.__tag.tables.role_attribute_tag[num]
  else
    a = "UNKNOWN"
  end
  return a
end

ltx.__tag.func.get_tag_from = __tag_get_tag_from

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

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

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

ltx.__tag.func.store_mc_data

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

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

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

ltx.__tag.func.store_mc_label

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

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

(End definition for ltx.__tag.func.store_mc_label)
ltx.__tag.func.store_mc_kid
This function is used in the traversing code. It stores a sub-chunk of a mc mcnum into the kids table.

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

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

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

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

3.2 Functions to insert the pdf literals

__tag_insert_emc_node
This insert the emc node.

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

(End definition for __tag_insert_emc_node)
__tag_insert_bmc_node
This inserts a simple bmc node

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

(End definition for __tag_insert_bmc_node)
This inserts a bdc node with a fix dict. TODO: check if this is still used, now that we create properties.

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

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

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

This is used to define a lua version of \pdffakespace

```lua
local function __tag_fakespace()
  tex.setattribute(lwspaceattributeid,1)
  tex.setattribute(lwfontattributeid,font.current())
end
```

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.

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

4 Function for the real space chars


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

}}
elseif id == MATH then
    inside_math = (n.subtype == 0)
end

return head
end

ltx.__tag.func.markspaceon=__tag_activate_mark_space
ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space

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

ltex.__tag.func.markspaceoff=__tag_deactivate_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

ltex.__tag.func.markspaceon=__tag_activate_mark_space

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

local function __tag_space_chars_shipout (box)
    local head = box.head
    if head then
        for n in node.traverse(head) do
            local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
            if n.id == HLIST then -- enter the hlist
                __tag_space_chars_shipout (n)
            elseif n.id == VLIST then -- enter the vlist
                __tag_space_chars_shipout (n)
            elseif n.id == GLUE then
                if ltx.__tag.trace.showspaces and spaceattr==1 then
                    __tag_show_spacemark (head,n,"0 1 0")
                end
            end
        end
    end
end

(End 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

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

ltx.__tag.func.markspaceoff=__tag_deactivate_mark_space

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

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

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

local function __tag_space_chars_shipout (box)
    local head = box.head
    if head then
        for n in node.traverse(head) do
            local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
            if n.id == HLIST then -- enter the hlist
                __tag_space_chars_shipout (n)
            elseif n.id == VLIST then -- enter the vlist
                __tag_space_chars_shipout (n)
            elseif n.id == GLUE then
                if ltx.__tag.trace.showspaces and spaceattr==1 then
                    __tag_show_spacemark (head,n,"0 1 0")
                end
            end
        end
    end
end

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

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

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

local function __tag_space_chars_shipout (box)
    local head = box.head
    if head then
        for n in node.traverse(head) do
            local spaceattr = nodegetattribute(n,iwspaceattributeid) or -1
            if n.id == HLIST then -- enter the hlist
                __tag_space_chars_shipout (n)
            elseif n.id == VLIST then -- enter the vlist
                __tag_space_chars_shipout (n)
            elseif n.id == GLUE then
                if ltx.__tag.trace.showspaces and spaceattr==1 then
                    __tag_show_spacemark (head,n,"0 1 0")
                end
            end
        end
    end
end

(End definition for __tag_activate_mark_space, ltx.__tag.func.markspaceon, and ltx.__tag.func.markspaceoff.)
if spaceattr==1 then
  local space
  local space_char = node.copy(default_space_char)
  local curfont = nodegetattribute(n, iwfontattributeid)
ltx.__tag.trace.log ("INFO SPACE-FUNCTION-FONT: ". tostring(curfont),3)
  if curfont and luaotfload.aux.slot_of_name(curfont,"space") then
    space_char.font=curfont
  end
  head, space = node.insert_before(head, n, space_char) --
  n.width = n.width - space.width
  space.attr = n.attr
end
end
end
end

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

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

5 Function for the tagging

ltx.__tag.func.mc_insert_kids

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

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

122
if single==1 then
    tex.sprint("null")
end
else
    ltx.__tag.trace.log("WARN TEX-MC-INSERT-MISSING: ".mcnum." doesn't exist",0)
end

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

---

ltx.__tag.func.store_struct_mcabs

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

function ltx.__tag.func.store_struct_mcabs (structnum,mcnum)
ltx.__tag.struct[structnum]=ltx.__tag.struct[structnum] or { }
ltx.__tag.struct[structnum]["mc"]=ltx.__tag.struct[structnum]["mc"] or { }
-- a structure can contain more than 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,"..structnum,3)
-- but every mc can only be in one structure
ltx.__tag.mc[mcnum]= ltx.__tag.mc[mcnum] or { }
ltx.__tag.mc[mcnum]["parent"] = structnum
end

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

ltx.__tag.func.store_mc_in_page

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

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

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

ltx.__tag.func.mark_page_elements

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

--[[
Now follows the core function
It wades through the shipout box and checks the attributes
ARGUMENTS
box: is a box,
mcpagecnt: num, the current page cnt of mc (should start at -1 in shipout box), needed for recursion
mccntprev: num, the attribute cnt of the previous node/whatever - if different we have a chunk border
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
name: string to describe the box
mctypeprev: num, the type attribute of the previous node/whatever

there are lots of logging messages currently. Should be cleaned up in due course.
]]--

123
One should also find ways to make the function shorter.

```
--]
function ltx.__tag.func.mark_page_elements (box,mcpagecnt,mcncntprev,mctopen,name,mctypeprev)
local name = name or ("SOMEBOX")
local mctypeprev = mctypeprev or -1
local abspage = status.total_pages + 1 -- the real counter is increased
-- inside the box so one off
-- if the callback is not used. (???)
ltx.__tag.trace.log ("INFO TAG-ABSPAGE: " .. abspage,3)
ltx.__tag.trace.log ("INFO TAG-ARGS: pagecnt".. mcpagecnt..
  " prev ".mcncntprev ..
  " type prev ".mctopen,4)
ltx.__tag.trace.log ("INFO TAG-TRAVERSING-BOX: ". tostring(name).. 
  " TYPE ". node.type(node.getid(box)),3)
local head = box.head -- ShipoutBox is a vlist?
if head then
  mcncnthead, mctopenhead,taghead = __tag_get_mc_cnt_type_tag (head)
ltx.__tag.trace.log ("INFO TAG-HEAD: ".
    node.type(node.getid(head))..
    " MC"..tostring(mcncnthead)..
    " => TAG " .. tostring(mctopenhead)..
    " => " .. 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)
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
    mcncnthead, mctopenhead,taghead = __tag_get_mc_cnt_type_tag (head)
ltx.__tag.trace.log ("INFO TAG-HEAD: ".
      node.type(node.getid(head))..
      " MC"..tostring(mcncnthead)..
      " => TAG " .. tostring(mctopenhead)..
      " => " .. tostring(taghead),3)
  else
    ltx.__tag.trace.log ("INFO TAG-NO-HEAD: head is ".
      tostring(head),3)
  end
  local mccnt, mctype, tag = __tag_get_mc_cnt_type_tag (n)
  local 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,mcncntprev,mctopenprev=
    ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcncntprev,mctopen,INTERNAL HLIST",mctypeprev)
  elseif n.id == VLIST then -- enter the vlist
    mcopen,mcpagecnt,mcncntprev,mctopenprev=
    ltx.__tag.func.mark_page_elements (n,mcpagecnt,mcncntprev,mctopen,INTERNAL VLIST",mctypeprev)
  elseif n.id == GLUE and not n.leader then -- at glue real space chars are inserted, but they
    -- been done if the previous shipout wandering, so here it
  elseif n.id == LOCAL_PAR then -- local_par is ignored
  elseif n.id == PENALTY then -- penalty is ignored
  elseif n.id == KERN then -- kern is ignored
  end
  ltx.__tag.trace.log ("INFO TAG-KERN-SUBTYPE: "..
    node.type(node.getid(n)).." ..n.subtype,4)
else
  -- math is currently only logged.
  -- we could mark the whole as math
  -- for inner processing the mlist_to_hlist callback is probably needed.
  if n.id == MATH then
```
ltx.__tag.trace.log("INFO TAG-MATH-SUBTYPE: ",
    node.type(node.getid(n)).." ".__tag_get_mathsubtype(n),4)
end
-- endmath
ltx.__tag.trace.log("INFO TAG-MC-COMPARE: current "..mccnt.." prev ",mccntprev,4)
if mccnt==mccntprev then -- a new mc chunk
ltx.__tag.trace.log("INFO TAG-NEW-MC-NODE: ",
    node.type(node.getid(n)).." MC"..tostring(mccnt).." <=> PREVIOUS ",tostring(mccntprev),4)
if mcopen==0 then -- there is a chunk open, close it (hope there is only one ... box.list=__tag_insert_emc_node (box.list,n)
mopen = 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]["artifact"]
        end
    end
else
    ltx.__tag.trace.log("INFO TAG-INSERT-TAG: ",
        tostring(tag),3)
    mcpagecnt = mcpagecnt +1
    ltx.__tag.trace.log("INFO TAG-INSERT-BDC: ",mcpagecnt,3)
    local dict= "/MCID "..mcpagecnt
    if ltx.__tag.mc[mccnt]["raw"] then
        ltx.__tag.trace.log("INFO TAG-USE-RAW: ",
            tostring(ltx.__tag.mc[mccnt]["raw"]),3)
        dict= dict .. " ",ltx.__tag.mc[mccnt]["raw"]
    end
if ltx.__tag.mc[mccnt]["alt"] then
    ltx.__tag.trace.log("INFO TAG-USE-ALT: ",
        tostring(ltx.__tag.mc[mccnt]["alt"]),3)
    dict= dict .. " ",ltx.__tag.mc[mccnt]["alt"]
end
if ltx.__tag.mc[mccnt]["actualtext"] then
    ltx.__tag.trace.log("INFO TAG-USE-ACTUALTEXT: ",
        tostring(ltx.__tag.mc[mccnt]["actualtext"])),3)
    dict= dict .. " ",ltx.__tag.mc[mccnt]["actualtext"]
end
box.list = __tag_insert_bdc_node (box.list,n,tag, dict)
ltx.__tag.func.store_mc_kid (mccnt,mcpagecnt,abspage)
ltx.__tag.func.store_mc_in_page(mccnt,mcpagecnt,abspage)
ltx.__tag.trace.show_mc_data (mccnt,3)
end
mcopen = mcopen + 1
else
    if tagunmarkedbool.mode == truebool.mode then
        ltx.__tag.trace.log("INFO TAG-NOT-TAGGED: this has not been tagged, using artifact",2)
        box.list = __tag_insert_bmc_node (box.list,n,"Artifact")
        mcopen = mcopen + 1
    else
        ltx.__tag.trace.log("WARN TAG-NOT-TAGGED: this has not been tagged",1)
    end
end
mccntprev = mccnt
end -- end 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) ..
    node.type(node.getid(box)),4)
return mcopen,mcpagecnt,mccntprev,mctypeprev
end

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

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

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

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 numentry ="
    local pdfpage = page-1
    if ltx.__tag.page[page] and ltx.__tag.page[page][0] then
        mcchunks=#ltx.__tag.page[page]
        ltx.__tag.trace.log("INFO PARENTTREE-NUM: page ".
            page.." has ".mcchunks.."+1 Elements ",4)
        for i=0,mcchunks do
            -- what does this log??
            ltx.__tag.trace.log("INFO PARENTTREE-CHUNKS: ..
                ltx.__tag.page[page][i],4)
        end
        if mcchunks == 0 then
            -- only one chunk so no need for an array
            local mcnum = ltx.__tag.page[page][0]
            local structnum = ltx.__tag.mc[mcnum]["parent"]
            --local objref = ltx.__tag.tables[proppname]["objref"] or "XXXX"
            local objref = __tag_pdf_object_ref(__tag/pdf/struct/'..structnum)
            ltx.__tag.trace.log("INFO PARENTTREE-STRUCT-OBJREF: =====>
                tostring(objref),5)
            numentry = pdfpage .. " [".. objref .. "]"
            ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page ".
                page.. " num entry = ".. numentry,3)
        else
            numentry = pdfpage .. " [" for i=0,mcchunks do
                local mcnum = ltx.__tag.page[page][i]
                local structnum = ltx.__tag.mc[mcnum]["parent"] or 0
                local proppname = "g__tag_struct_"..structnum.."_prop"
                --local objref = ltx.__tag.tables[proppname]["objref"] or "XXXX"
                local objref = __tag_pdf_object_ref(__tag/pdf/struct/'..structnum)
                numentry = numentry .. " ". objref end
            numentry = numentry .. "] "
            ltx.__tag.trace.log("INFO PARENTTREE-NUMENTRY: page ".
                page.. " num entry = ".. numentry,3)
        end
    else
        ltx.__tag.trace.log("INFO PARENTTREE-NO-DATA: page ".page,3)
    end
    return numentry
```

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

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

(/lua)
Part IX

The **tagpdf-roles** module

Tags, roles and namespace code

Part of the tagpdf package

---

The `add-new-tag` key can be used in `\tagpdfsetup` to declare and rolemap new tags. It takes as value a key-value list or a simple `new-tag/old-tag`.

The key-value list knows the following keys:

- **tag** This is the name of the new tag as it should then be used in `\tagstructbegin`.
- **namespace** This is the namespace of the new tag. The value should be a shorthand of a namespace. The allowed values are currently `pdf`, `pdf2`, `mathml`, `latex`, `latex-book`, and `user`. The default value (and recommended value for a new tag) is `user`. The public name of the user namespace is `tag/NS/user`. This can be used to reference the namespace e.g. in attributes.
- **role** This is the tag the tag should be mapped too. In a PDF 1.7 or earlier this is normally a tag from the `pdf` set, in PDF 2.0 from the `pdf`, `pdf2` and `mathml` set. It can also be a user tag. The tag must be declared before, as the code retrieves the class of the new tag from it. The PDF format allows mapping to be done transitively. But tagpdf can’t/won’t check such unusual role mapping.
- **role-namespace** If the role is a known tag the default value is the default namespace of this tag. With this key a specific namespace can be forced.

Namespaces are mostly a PDF 2.0 property, but it doesn’t harm to set them also in a PDF 1.7 or earlier.

---

1 Code related to roles and structure names

1.1 Variables

Tags are used in structures (`\tagstructbegin`) and mc-chunks (`\tagmcbegin`).

---

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.

\g__tag_role_tags_NS_prop This is the core list of tag names. It uses tags as keys and the shorthand (e.g. pdf2, or mathml) of the default name space as value. We store the default name space also in pdf <2.0, even if not needed: it doesn’t harm and simplifies the code. There is no need to access this from lua, so we use the standard prop commands.

: \prop_new:N \g__tag_role_tags_NS_prop
(End definition for \g__tag_role_tags_NS_prop.)

\g__tag_role_tags_class_prop
With pdf 2.0 we store the class in the NS 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.
\prop_new:N \g__tag_role_tags_class_prop
(End definition for \g__tag_role_tags_class_prop.)

\g__tag_role_NS_prop
This holds the list of supported name spaces. The keys are the name tagpdf will use, the
values the object reference. The urls identifier are stored in related dict object.

mathml http://www.w3.org/1998/Math/MathML
pdf2 http://iso.org/pdf2/ssn
pdf http://iso.org/pdf/ssn (default)
user \c__tag_role_userNS_id_str (random id, for user tags)
latex https://www.latex-project.org/ns/dflt/2022
latex-inline https://www.latex-project.org/ns/inline/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.
\prop_new:N \g__tag_role_NS_prop
(End definition for \g__tag_role_NS_prop.)

\g__tag_role_index_prop
This prop contains the standard tags (pdf in pdf<2.0, pdf,pdf2 + mathml in pdf 2.0) as
keys, the values are a two-digit number. These numbers are used to get the containment
rule of two tags from the intarray.
\prop_new:N \g__tag_role_index_prop
(End definition for \g__tag_role_index_prop.)

\l__tag_role_debug_prop
This variable is used to pass more infos to debug messages.
\prop_new:N \l__tag_role_debug_prop
(End definition for \l__tag_role_debug_prop.)

We need also a bunch of temporary variables.
\l__tag_role_tag_tmpa_tl
\l__tag_role_tag_namespace_tmpa_tl
\l__tag_role_role_tmpa_tl
\l__tag_role_role_namespace_tmpa_tl
\l__tag_role_tmpa_seq
\tl_new:N \l__tag_role_tag_tmpa_tl
\tl_new:N \l__tag_role_tag_namespace_tmpa_tl
\tl_new:N \l__tag_role_role_tmpa_tl
\tl_new:N \l__tag_role_role_namespace_tmpa_tl
\seq_new:N \l__tag_role_tmpa_seq
(End definition for \l__tag_role_tag_tmpa_tl and others.)
1.2 Namespaces

The following commands setups a name space. With pdf version <2.0 this is only a
prop with the rolemap. With pdf 2.0 a dictionary must be set up. Such a name space
dictionaries can contain an optional /Schema and /RoleMapNS entry. We only reserve the
objects but delay the writing to the finish code, where we can test if the keys and the
name spaces are actually needed. This commands setups objects for the name space and
its rolemap. It also initialize a dict to collect the rolemaps if needed, and a property with
the tags of the name space and their rolemapping for loops. It is unclear if a reference
to a schema file will be ever needed, but it doesn’t harm.

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

(End definition for g__tag_role/RoleMap_dict and \g__tag_role_rolemap_prop.)

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

\__tag_role_NS_new:nnn

\pdf_version_compare:NnTF < {2.0}
{\cs_new_protected:Npn \__tag_role_NS_new:nnn #1 #2 #3 }
{\prop_new:c { g__tag_role_NS_#1_prop }\prop_new:c { g__tag_role_NS_#1_class_prop }\prop_gput:Nnx \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}{Type}{/Namespace}\pdf_string_from_unicode:nnN{utf8/string}{#2}\l__tag_tmpa_str \tl_if_empty:NF \l__tag_tmpa_str
{ \pdfdict_gput:nx {g__tag_role/Namespace_#1_dict}{NS}{\l__tag_tmpa_str} }
}\RoleMapNS is added in tree \tl_if_empty:nF (#3)
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:Nx \c__tag_role_userNS_id_str { data:, \int_to_Hex:n{\int_rand:n{65535}} \int_to_Hex:n{\int_rand:n{65535}} - \int_to_Hex:n{\int_rand:n{65535}} \int_to_Hex:n{\int_rand:n{65535}} - \int_to_Hex:n{\int_rand:n{65535}} \int_to_Hex:n{\int_rand:n{65535}} - \int_to_Hex:n{\int_rand:n{65535}} \int_to_Hex:n{\int_rand:n{16777215}} \int_to_Hex:n{\int_rand:n{16777215}}}

(End definition for \c__tag_role_userNS_id_str.)

Now we setup the standard names spaces. The mathml space is currently only loaded for pdf 2.0.

\__tag_role_NS_new:nnn {pdf} {http://iso.org/pdf/ssn}{}
\__tag_role_NS_new:nnn {pdf2} {http://iso.org/pdf2/ssn}{}
\pdf_version_compare:NnF < {2.0}

1.3 Adding a new tag
Both when reading the files and when setting up a tag manually we have to store data in various places.

\__tag_role_alloctag:nnn This command allocates a new tag without role mapping. In the lua backend it will also record the attribute value.
\pdf_version_compare:NnTF < {2.0}
\{
\sys_if_engine_luatex:TF
{
\cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
{
  \lua_now:e { ltx.__tag.func.alloctag ('#1') }
  \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
  \prop_gput:cn {g__tag_role_NS_#2_prop} {#3}
} \prop_gput:cn {g__tag_role_NS_#2_class_prop} {#4}{--UNUSED--}
}

{\sys_if_engine_luatex:TF
\cs_new_protected:Npn \__tag_role_alloctag:nnn #1 #2 #3 %#1 tagname, ns, type
{
  \lua_now:e { ltx.__tag.func.alloctag ('#1') }
  \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
  \prop_gput:cn {g__tag_role_NS_#2_prop} {#3}
} \prop_gput:cn {g__tag_role_NS_#2_class_prop} {#4}{--UNUSED--}
}

{\sys_if_engine_luatex:TF
\cs_new_protected:Npn \__tag_role_add_tag:nn % (new) name, reference to old
{
  \lua_now:e { ltx.__tag.func.addtag ('#1') }
  \prop_gput:Nnn \g__tag_role_tags_NS_prop {#1}{#2}
  \prop_gput:cn {g__tag_role_NS_#2_prop} {#3}
} \prop_gput:cn {g__tag_role_NS_#2_class_prop} {#4}{--UNUSED--}
}

\cs_generate_variant:Nn \__tag_role_alloctag:nnn {nnV}
(End 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
{

134
checks and messages
\_\_tag_check_add_tag_role:nn (#1){#2}
\prop_if_in:NnF \g__tag_role_tags_NS_prop (#1)
{\int_compare:nNnT {\l__tag_loglevel_int} > { 0 } }
\msg_info:nnn { tag }{new-tag}{#1}
\prop_get:NnN \g__tag_role_tags_class_prop {#2}\l__tag_tmpa_tl
\quark_if_no_value:NT \l__tag_tmpa_tl
{\tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}}
\__tag_role_alloctag:nnV (#1){user}\l__tag_tmpa_tl

We resolve rolemapping recursively so that all targets are stored as standard tags.
\l__tag_tmpa_tl\prop_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
\quark_if_no_value:NTF \l__tag_tmpa_tl
{\prop_gput:Nnx \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 definition for \__tag_role_add_tag:nnn)

\_\_tag_role_add_tag:nnnn

The pdf 2.0 version takes four arguments: tag/namespace/role/namespace
\cs_new_protected:Nn \_\_tag_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_get:cnN { g__tag_role_NS_#4_class_prop } {#3}\l__tag_tmpa_tl
\quark_if_no_value:NT \l__tag_tmpa_tl
{\tl_set:Nn\l__tag_tmpa_tl{--UNKNOWN--}}
\__tag_role_alloctag:nnV (#1){#2}\l__tag_tmpa_tl
\pdfdict_gput:nnx {g__tag_role/RoleMapNS_#2_dict}{#1}
{\pdf_name_from_unicode_e:n{#3}
We resolve rolemapping recursively so that all targets are stored as standard tags for the tests.

\tl_if_empty:nF { #2 }
\prop_get:cnN { g__tag_role_NS_#4_prop } {#3}\l__tag_tmpa_tl
\quark_if_no_value:NTF \l__tag_tmpa_tl
\prop_gput:cnx { g__tag_role_NS_#2_prop } {#1}{\tl_to_str:n{#3}\tl_to_str:n{#4}}
\prop_gput:cno { g__tag_role_NS_#2_prop } {#1}{\l__tag_tmpa_tl}
\cs_generate_variant:Nn \__tag_role_add_tag:nnnn {VVVV}

(End definition for \__tag_role_add_tag: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 name space. The definition differ for pdf 2.0. as we have proper name spaces there. With pdf<2.0 not special name spaces shouldn’t update the default role or add to the rolemap again. We use a boolean here.

\bool_new:N \l__tag_role_update_bool
\bool_set_true:N \l__tag_role_update_bool
\pdf_version_compare:NnTF < {2.0} {
\cs_new_protected:Npn \__tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop %
% #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
%
\tl_if_empty:nF { #2 }
\bool_if:NTF \l__tag_role_update_bool {
\tl_if_empty:nF {#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--}
}
\l__tag_tmpa_tl{\l__tag_tmpa_tl}
\tl_set:Nn \l__tag_tmpa_tl {#5}
}
\_\_\_tag_role_alloctag:nnV \{#2}\{#1\}l\_tag tmpa t1
\tl_if_empty:nF \{#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\_prop\} \{#2\}\{#3\}\{}
\prop_gput:cnn \{g\_\_\_tag_role_NS_#1\_class\_prop\} \{#2\}{--UNUSED--}
\}
\}
\}
\}
\cs_new_protected:Npn \_\_\_tag_role_read_namespace_line:nw #1#2,#3,#4,#5,#6\q_stop
% #1 NS, #2 tag, #3 rolemap, #4 NS rolemap #5 type
\{
\tl_if_empty:nF \{#2\}
\{
\tl_if_empty:nTF \{#5\}
\{
\prop_get:cnN \{g\_\_\_tag_role_NS_#4\_class\_prop\} \{#3\}l\_tag tmpa t1
\quark_if_no_value:NT \l\_tag tmpa t1
\{
\l\_\_\_set:Nn \l\_tag tmpa t1{--UNKNOWN--}
\}
\}
\\tl_set:Nn \l\_tag tmpa t1 \{#5\}
\}
\_\_\_tag_role_alloctag:nnV \{#2\}\{#1\}l\_tag tmpa t1
\bool_lazy_and:nnT \{ ! \tl_if_empty_p:n \{#3\} \} ! \str_if_eq_p:nn \{#1\} {pdf2} \}
\{
\_\_\_tag_role_add_tag:nnnn \{#2\}\{#1\}\{#3\}\{#4\}
\}
\}
\}
\(\)End definition for \_\_\_tag_role_read_namespace_line:nw\)

\_\_\_tag_role_read_namespace:n This command reads the namespace file.
\cs_new_protected:Npn \_\_\_tag_role_read_namespace:n \#1 \ename of namespace
\{
\prop_if_exist:cF \{g\_\_\_tag_role_NS_#1\_prop\}
\{ \msg_warning:nnn \{tag\} \{namespace-unknown\}\{#1\} \}
\file_if_exist:nTF \{ tagpdf-ns-#1.def\}
\{
\ior_open:Nn \g\_tmpa_iork \{tagpdf-ns-#1.def\}
\msg_info:nnn \{tag\} \{read-namespace\}\{#1\}
\ior_map_inline:Nn \g\_tmpa_iork
1.5 Reading the default data

The order is important as we want pdf2 and latex as default.

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

(End definition for \__tag_role_read_namespace:n.)
1.6 Parent-child rules

PDF defines 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.

\begin{verbatim}
\intarray_new:Nn \g__tag_role_parent_child_intarray {6000}
\end{verbatim}

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

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

The helper command is used to store the rule. It assumes that parent and child are given as 2-digit number!

\begin{verbatim}
\int_zero:N \l__tag_tmpa_int
\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} }
\ior_map_inline:Nn \g_tmpa_ior {\l__tag_tmpa_seq}
\int_compare:nNnTF \l__tag_tmpa_int = 1 {\prop_item:Nn\c__tag_role_rules_prop\l__tag_tmpa_int} \l__tag_tmpa_int
\end{verbatim}

Now the main loop over the file

\begin{verbatim}
\ior_map_inline:Nn \g_tmpa_ior {\tl_if_empty:nF{#1} \int_incr:N\l__tag_tmpa_int \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 } \int_compare:nNnTF \l__tag_tmpa_int=1}
\end{verbatim}

1.6.1 Reading in the csv-files

This counter will be used to identify the first (non-comment) line

\begin{verbatim}
\int_zero:N \l__tag_tmpa_int
\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} }
\ior_map_inline:Nn \g_tmpa_ior {\l__tag_tmpa_seq}
\int_compare:nNnTF \l__tag_tmpa_int = 1 {\prop_item:Nn\c__tag_role_rules_prop\l__tag_tmpa_int} \l__tag_tmpa_int
\end{verbatim}

Now the main loop over the file

\begin{verbatim}
\ior_map_inline:Nn \g_tmpa_ior {\tl_if_empty:nF{#1} \int_incr:N\l__tag_tmpa_int \seq_set_from_clist:Nn\l__tag_tmpa_seq { #1 } \int_compare:nNnTF \l__tag_tmpa_int=1}
\end{verbatim}

139
This handles the header line. It gives the tags 2-digit numbers

\seq_map_indexed_inline:Nn \l__tag_tmpa_seq
\prop_gput:Nnx\g__tag_role_index_prop
\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:Nnx
\__tag_store_parent_child_rule:nnn {##1}{\l__tag_tmpb_tl}{ ##2 }
}

close the read handle.
\ior_close:N \g_tmpa_ior

The Root, Hn and mathml tags are special and need to be added explicitly
\prop_get:NnN \g__tag_role_index_prop {StructTreeRoot} \l__tag_tmpa_tl
\prop_gput:Nnx \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:Nnx \g__tag_role_index_prop {H#1} \l__tag_tmpa_tl } \int_step_inline:nn{10} { \prop_gput:Nnx \g__tag_role_index_prop {H#1} \l__tag_tmpa_tl } }
all mathml tags are currently handled identically

\prop_get:Nn\g__tag_role_index_prop {mathml}\l__tag_tmpa_tl
\prop_get:Nn\g__tag_role_index_prop {math}\l__tag_tmpb_tl
\prop_map_inline:Nn \g__tag_role_NS_mathml_prop
{
  \prop_gput:NN\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
}
\prop_gput:NN\g__tag_role_index_prop{math}\l__tag_tmpb_tl

1.6.2 Retrieving the parent-child rule

Part, Div and NonStruct have no own rules, instead the parent(s) have to be inspected.
To store this real parent we use this tlvra
\tl_new:N \l__tag_role_real_parent_tl
(End definition for \l__tag_role_real_parent_tl.)

\tag_role_get_parent_child_rule:nN
This command retrieves the rule (as a number) and stores it in the tl-var. TODO check temporary variables. Check if the tl-var should be fix. The arguments should be standard tags for which a rule exist and role mapping should have already be done.
\tl_new:N \l__tag_parent_child_check_tl
\cs_new_protected:Npn \__tag_role_get_parent_child_rule:nnN #1 #2 #3
% #1 parent (string) #2 child (string) #3 tl for state
{
\tl_set:Nn \l__tag_role_real_parent_tl {#1}
\prop_get:Nn\g__tag_role_index_prop{#1}\l__tag_tmpa_tl
\prop_get:Nn\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:Nx#3
{\intarray_item:Nn \g__tag_role_parent_child_intarray \l__tag_tmpa_tl\l__tag_tmpb_tl}
}
If the state is we have to check the parents from the stack and use the first which is not Part, Div or NonStruct
\int_compare:nNnT
{#3}= {\prop_item:Nn\c__tag_role_rules_prop{}}
{
\seq_set_eq:NN \l__tag_role_tmpa_seq \g__tag_struct_tag_stack_seq
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.
\seq_pop_left:NN \l__tag_role_tmpa_seq \l__tag_get_tmpc_tl
\seq_map_inline:Nn\l__tag_role_tmpa_seq
{\tl_if_in:nnF {-Part-Div-NonStruct-}{-##1-}}
This is the message, this can perhaps go into debug mode.
\__tag_check_parent_child:nnnnN
This is the main command. It has to retrieve the standard tags for a comparison. 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
{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:NnNF \g__tag_role_rolemap_prop {#1}\l__tag_tmpa_tl
{\tl_set:Nn \l__tag_tmpa_tl {\q_no_value}}}
now the child
 \prop_get:NnTF \g__tag_role_index_prop {#3}\l__tag_tmpb_tl
{\tl_set:Nn \l__tag_tmpb_tl {#3}}
{\prop_get:NnNF \g__tag_role_rolemap_prop {#3}\l__tag_tmpb_tl
{\tl_set:Nn \l__tag_tmpb_tl {\q_no_value}}} 
if we got tags for parent and child we call the checking command 
\bool_lazy_and:nnTF
{! \quark_if_no_value_p:N \l__tag_tmpa_tl }
{! \quark_if_no_value_p:N \l__tag_tmpb_tl }

143
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.

\cs_new_protected:Npn \__tag_check_parent_child:nnnnN #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}
\msg_warning:nxxx { tag } {role-parent-child} { #1 } { #2 } { unknown! }
\tl_set:Nn {0}
\tl_clear:N \l__tag_role_tag_namespace_tmpb_tl
\msg_warning:nxxx { tag } {role-parent-child} { #1 } { #2 } { unknown! }
}\__tag_check_parent_child:nnnN {#1}{#2}{#3}
\tl_set:Nn #5 {0}
\msg_warning:nxxx { tag } {role-parent-child} { #1 }
\tl_clear:N \l__tag_role_tag_namespace_tmpa_tl
\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 {#1}\l__tag_role_tag_namespace_tmpa_tl {#2}\l__tag_role_tag_namespace_tmpb_tl { unknown! }
\tl_set:Nn {0}
\msg_warning:nxxx { tag } {role-parent-child} { #1 } { #2 } { unknown! }
\__tag_check_parent_child:VVN \l__tag_tmpa_tl \l__tag_tmpb_tl #5
\tl_set:Nn #5 {0}
\msg_warning:nxxx { tag } {role-parent-child} { #1 }
\tl_clear:N \l__tag_role_tag_namespace_tmpa_tl
\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 {#1}\l__tag_role_tag_namespace_tmpa_tl {#2}\l__tag_role_tag_namespace_tmpb_tl { unknown! }
\tl_set:Nn {0}
\msg_warning:nxxx { tag } {role-parent-child} { #1 }
\tl_clear:N \l__tag_role_tag_namespace_tmpa_tl
\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 {#1}\l__tag_role_tag_namespace_tmpa_tl {#2}\l__tag_role_tag_namespace_tmpb_tl
If the namespace is empty, we assume a standard tag, otherwise we retrieve the role-mapping from the namespace

```latex
\tl_if_empty:nTF {#2}
\l_set:Nn \l__tag_tmpa_tl {#1}
\prop_get:cnNTF
\l__tag_role_NS_#2_prop
\l__tag_tmpa_tl
\tl_set:Nx \l__tag_tmpa_tl \{\tl_head:N\l__tag_tmpa_tl\}
\tl_if_empty:NT\l__tag_tmpa_tl
\l_set:Nn \l__tag_tmpa_tl {#1}
\}
\l_set:Nn \l__tag_tmpa_tl {\tl_head:N\l__tag_tmpa_tl}
\tl_if_empty:NT\l__tag_tmpa_tl
\l_set:Nn \l__tag_tmpa_tl {#1}
\}
\l_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 role-mapping from the namespace

```latex
\tl_if_empty:nTF {#4}
\l_set:Nn \l__tag_tmpb_tl {#3}
\prop_get:cnNTF
\l__tag_role_NS_#4_prop
\l__tag_tmpb_tl
\tl_set:Nx \l__tag_tmpb_tl \{\tl_head:N\l__tag_tmpb_tl\}
\tl_if_empty:NT\l__tag_tmpb_tl
\l_set:Nn \l__tag_tmpb_tl {#3}
\}
\l_set:Nn \l__tag_tmpb_tl {\q_no_value}
\}
```

and now get the relation

```latex
\bool_lazy_and:nnTF
\tl_if_empty:nTF \q_no_value_p:N \l__tag_tmpa_tl
\tl_if_empty:nTF \q_no_value_p:N \l__tag_tmpb_tl
\__tag_role_get_parent_child_rule:VVN \l__tag_tmpa_tl \l__tag_tmpb_tl #5
\}
```

145
1.7 Remapping of tags

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

The following command provide 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.

\begin{verbatim}
\l__tag_role_remap_tag_tl
\l__tag_role_remap_NS_tl
\tl_new:N \l__tag_role_remap_tag_tl
\tl_new:N \l__tag_role_remap_NS_tl
\end{verbatim}

\texttt{\l__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?

\begin{verbatim}
\cs_new_protected:Npn \__tag_role_remap: { }
\end{verbatim}

\texttt{\_tag_role_remap_id:} This is copy in case we have to restore the main command.

\begin{verbatim}
\cs_set_eq:NN \__tag_role_remap_id: \__tag_role_remap: 
\end{verbatim}

\texttt{\_tag_role_remap_inline:} The mapping is meant to “degrade” tags, e.g. if used inside some complex object. The pdf<2.0 code maps the tag to the new role, the pdf 2.0 code only switch the NS.

\begin{verbatim}
\pdf_version_compare:NnTF < {2.0} {
\cs_new_protected:Npn \_tag_role_remap_inline:
{
\prop_get:cVNT { g__tag_role_NS_latex-inline_prop }\l__tag_role_remap_tag_tl\l__tag_tmpa_tl
\tl_set:Nx\l__tag_role_remap_tag_tl
{
\exp_last_unbraced:NV\use_i:nn \l__tag_tmpa_tl
}
\tl_set:Nx\l__tag_role_remap_NS_tl

\end{verbatim}
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 }
  { ,tag .tl_set:N = \l__tag_role_tag_tmpa_tl
    ,tag-namespace .tl_set:N = \l__tag_role_tag_namespace_tmpa_tl
    ,role .tl_set:N = \l__tag_role_role_tmpa_tl
    ,role-namespace .tl_set:N = \l__tag_role_role_namespace_tmpa_tl }
\keys_define:nn { __tag / setup }
  { add-new-tag .code:n =
    \keys_set_known:nnnnN
    { __tag/tag-role }
    { tag-namespace=user,
      role-namespace=, %so that we can test for it.
      #1 }
    { \l__tag_role_tag_tmpa_tl
      \tl_if_empty:NF \l_tmpa_tl
      { ...
```
(End definition for tag (rolemap-key) and others. These functions are documented on page 129.)
Part X

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

interwordspace\{setup-key\} This key allows to activate/deactivate the real space chars if the engine supports it. The allowed values are true, on, false, off.

show-spaces\{setup-key\} 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.

1 Code for interword spaces

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

interwordspace\{setup-key\}

\keys_define:nn { __tag / setup }
{ interwordspace .choices:nn = { true, on } \mag_warning:nx {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
interwordspace .choices:nn = { false, off } \mag_warning:nx {tag}{sys-no-interwordspace}{\c_sys_engine_str} },
interwordspace .default:n = true,
show-spaces .bool_set:N = \l__tag_showspaces_bool
}
\sys_if_engine_pdftex:T
{ \pdfglyphtounicode{space}{0020}
\keys_define:nn { __tag / setup }
{ interwordspace .choices:nn = { true, on } \pdffinterwordspaceon },
interwordspace .choices:nn = { false, off }\pdffinterwordspaceon },
interwordspace .default:n = true,
\keys_define:nn { __tag / setup }
{
  interwordspace .choices:nn = { true, on, false, off }
  \msg_warning:nnn {tag}{sys-no-interwordspace}{dvi},
  interwordspace .default:n = true,
  show-spaces .bool_set:N = \l__tag_showspaces_bool
}
\keys_define:nn { __tag / setup }
{
  interwordspace .choices:nn = { true, on }
  \bool_gset_true:N \g__tag_active_space_bool
  \lua_now:e{ltx.__tag.func.markspaceon()}
  \interwordspace .choices:nn = { false, off }
  \bool_gset_false:N \g__tag_active_space_bool
  \lua_now:e{ltx.__tag.func.markspaceoff()}
  \interwordspace .default:n = true,
  show-spaces .choice:,
  show-spaces / true .code:n =
  {\lua_now:e{ltx.__tag.trace.showspaces=true}},
  show-spaces / false .code:n =
  {\lua_now:e{ltx.__tag.trace.showspaces=nil}},
  show-spaces .default:n = true
}
\sys_if_engine_luatex:T
{
  \keys_define:nn { __tag / setup }
  {
    \interwordspace .choices:nn =
    \bool_gset_true:N \g__tag_active_space_bool
    \lua_now:e{ltx.__tag.func.markspaceon()}
    \interwordspace .choices:nn =
    \bool_gset_false:N \g__tag_active_space_bool
    \lua_now:e{ltx.__tag.func.markspaceoff()}
    \interwordspace .default:n = true,
    show-spaces .choice:,
    show-spaces / true .code:n =
    {\lua_now:e{ltx.__tag.trace.showspaces=true}},
    show-spaces / false .code:n =
    {\lua_now:e{ltx.__tag.trace.showspaces=nil}},
    show-spaces .default:n = true
  }
}

(End definition for interwordspace (setup-key) and show-spaces (setup-key). These functions are documented on page 149.)

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

(End definition for \_\_tag_fakespace:)
Index

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

Symbols
\  10, 23, 37, 38, 39, 47
\u  240, 254

A
activate, (setup-key)  32, 182
activate-all, (setup-key)  6, 228
activate-mc, (setup-key)  6, 228
activate-space, (setup-key)  6, 228
activate-struct, (setup-key)  6, 228
activate-tree, (setup-key)  6, 228
actual-text, (mc-key)  56, 221, 413
actual-text, (struct-key)  84, 371
add-new-tag, (setup-key)  129, 643
\AddToHook  13, 16, 50, 79, 197, 232, 246, 260, 271, 309
AF, (struct-key)  85, 479
AFinline, (struct-key)  85, 479
AFinline-o, (struct-key)  85, 479
alt, (mc-key)  56, 221, 413
alt, (struct-key)  84, 371
artifact, (mc-key)  56, 221, 413
artifact-bool internal commands:
  _artifact-bool  113
artifact-type internal commands:
  _artifact-type  113
attr, (struct-key)  85, 570
attribute, (struct-key)  85, 536

B
bool commands:
  \bool_gset_eq:NN  329, 342, 354, 370
  \bool_gset_false:N  39, 54, 213, 330, 355, 401
  \bool_gset_true:N  38, 48, 119, 160, 336
  \bool_if:NTF  9, 18, 28, 32, 33, 37, 73, 155, 175, 183, 202, 212, 224, 234, 238, 248, 252, 255, 269, 270, 273, 279, 321, 324, 337, 340, 349, 357, 365, 638
  \bool_if:NTF  6, 299
  \bool_lazy_all:nnF  72, 199
  \bool_lazy_and:nnF  89, 99, 247, 378, 386, 427, 444, 496, 523, 586
  \bool_lazy_and:p:nn  8
  \bool_new:N  11, 15, 16, 37, 61, 114, 115, 116, 117, 118, 120, 122, 124, 193, 217, 218, 320
  \bool_set_false:N  161, 185, 186, 191, 192, 204, 205, 214, 279, 301, 323, 348
  \bool_set_true:N  121, 123, 194, 196, 197, 217, 218, 282, 300
box commands:
  \box_dp:N  177, 181
  \box_ht:N  167
  \box_new:N  109, 110
  \box_set_dp:NN  175, 177
  \box_set_eq:NN  190
  \box_set_ht:NN  174, 176
  \box_use_drop:N  179, 183
  \boxmaxdepth  67, 178

C
\c  203, 204
\c@g internal commands:
  \c@g__tag_MDIC_abs_int  9, 25, 34, 47, 54, 65, 71, 73, 135, 148, 163, 237, 242, 271, 311, 376
  \c@g__tag_parenttree_obj_int  71
  \c@g__tag_struct_abs_int  6, 18, 68, 137, 140, 142, 383, 408, 420, 432, 445, 452, 464, 474, 505, 507, 512, 523, 527, 528, 530, 531, 533, 538, 547, 551, 552, 554, 555, 557, 562, 592, 602, 603, 604, 605, 608, 610, 616, 619, 634, 636, 665, 670, 761, 863, 866, 912
\cc  132
\chapter  138, 290, 302
clist commands:
  \clist_const:NN  111, 112
  \clist_map_inline:nn  135, 459
  \clist_new:NN  107
  \clist_set:NN  840, 874
color commands:
  \color_select:n  240, 254
cs commands:
  \cs_gset_eq:NN  224, 662, 663, 758, 759
  \cs_if_exist:NTF  81, 275, 290, 302, 311

152
\cs_if_exist_p:N \cs_if_exist_use:NTF \cs_if_free:NTF \cs_new:Nn \cs_new:Npn \cs_new_protected:Nn \cs_new_protected:Npn \cs_set:Nn \cs_set:Npn \cs_set_eq:NN \cs_set_protected:Nn \cs_set_protected:Npn \cs_to_str:N \DeclareOption \dim commands: \c_max_dim \c_zero_dim \documentclass \DocumentMetadata 
\endinput \PackageWarning \PackageError \errormessage \PackageError \errormessage \PackageWarning \PackageWarning \PackageWarning \PackageWarning 

\exp_args:Nee \exp_args:NNno \exp_args:NNnx \exp_args:Nnx \exp_args:NV \exp_args:Nx \exp_last_unbraced:NV \exp_not:n 
\file_if_exist:nTF \file_input:n \fontencoding \fontfamily \fontseries \fontshape \fontsize \footins 
\group_begin: \group_end: 
\hbox_set:Nn \hbox \hook_gput_code:nnn \hook_new:n \hook_use:n 
\ignorespaces \int commands: \int_case:nnTF \int_compare:nNnTF \int_compare:nTF \int_compare_p:nNn \int_eval:n 
\ExecuteOptions \exp commands: \exp_args:N \exp_args:Ne 

F \file commands: \file_if_exist:nTF \file_input:n \fontencoding \fontfamily \fontseries \fontshape \fontsize \footins 

G group commands: \group_begin: \group_end: 

H \hbox commands: \hbox_set:Nn \hook \hook_new:n \hook_use:n 

I \ignorespaces \int commands: \int_case:nnTF \int_compare:nNnTF \int_compare:nTF \int_compare_p:nNn \int_eval:n 

\endinput
\prg_new_conditional:Nnn ...... 59, 222
\prg_new_conditional:Npn ...... 
                        66, 87, 97, 291, 297, 308
\prg_new_eq_conditional:NNn . 73, 229
\prg_return_false: .......... 67,
                        69, 84, 94, 104, 226, 294, 306, 312
\prg_return_true: ............ ...
                        70, 81, 91, 101, 225, 295, 305, 311
\prg_set_conditional:Npn ...... 70
\ProcessOptions ............ 41
\prop_commands:            
\prop_clear:N ............ 95
\prop_count:N .......... 116
\prop_get:NNn ........... 139,
                        147, 165, 181, 206, 237, 343, 355,
                        357, 370, 371, 384, 385, 520, 521, 790
\prop_get:NNnTF .......... 93, 118, 121,
                        135, 137, 152, 171, 247, 416, 476,
                        481, 486, 491, 551, 571, 613, 633, 670
\prop_gput:NNn ........... 214
\prop_gremove:N ........... 290
\prop_if_exist:NNTF ....... 258, 728
\prop_if_exist_p:N ....... 380
\prop_if_in:NNTF .......... 66, 109,
                        117, 132, 216, 282, 675, 848, 886, 890
\prop_if_in_p:N ........... 429, 446
\prop_item:N ...... 34, 70, 105, 131,
                        166, 172, 200, 289, 292, 315, 361,
                        395, 397, 432, 435, 449, 452, 895, 902
\prop_map_inline:N ........ 176, 230, 292, 304, 372
\prop_map_tokens:N ........ 248
\prop_new:N ...... 7,
                        8, 9, 10, 11, 15, 18, 23, 24,
                        31, 32, 62, 63, 104, 167, 199, 819, 822
\prop_put:NNn ........... 102, 130, 474, 475, 544, 545
\prop_show:N ........... 58, 92, 174, 681, 684, 866, 891
\ProvidesExplFile ......... 3
\ProvidesExplPackage ...... 3,
                        3, 3, 3, 3, 3, 3, 3, 7, 26, 46, 815
\quad ............... 162, 163
quark commands:          
\q_no_value ........ 483, 493, 563, 583
\quark_if_no_value:NTF .... 140, 148, 166, 182, 207, 215, 238, 797
\quark_if_no_value:p:N .... 387, 388, 497, 498, 524, 525, 587, 588
\q_stop .................. 197, 230, 266
R
rax_i(nc-key) ............ 56, 221, 413
ref_i(struct-key) ........ 85, 371
ref commands:            
\ref_attribute_gset:nnnn .... 
                        ................. 136, 138, 145, 147, 149
\ref_label:nn .......... 132, 154, 332
\ref_value:nn .......... 84, 462
\ref_value:nnn . 6, 81, 81, 83, 160, 165
ref internal commands:   
\_\_\_ref_value:nnnn .... 86, 89
regex commands:          
\regex_replace_once:nnn .... 
                        ................. 202
\renewcommand .......... 300, 301
\RenewDocumentCommand .... 8
\RequirePackage ........ 20, 42, 43, 272, 275, 281, 284
\rlap .................. 254
role_i(role-map-key) ...... 129, 643
role-missing ............ 18, 43
role-\_namespace_i(role-map-key) .. 129, 643
role-parent-child ........ 46, 48
role-tag ................ 18, 50
role-unknown ............ 18, 43
role-unknown-tag ........ 18, 43
root-AF_i(setup-key) ...... 86, 569
S
\selectfont .............. 6
seq commands:            
\seq_clear:N ........... 214, 258
\seq_const_from_clist:Nn ... 21, 34
\seq_count:N ........... 6, 22, 25, 226, 856, 858, 860, 882, 908
\seq_get:NNTF .......... 324, 627, 706, 713
\seq_gpop:NN ........... 699
\seq_gpop:NNTF .......... 97, 700
\seq_gpop_left:NN ........ 201
\seq_gpush:Nn ........... 12, 14, 80, 87, 634, 635
\seq_gput_left:Nn ...... 206, 852
\seq_gput_left:NN ...... 32, 134, 170, 278
\seq_gremove_duplicates:N .... 195
\seq_gset_eq:NN ........ 156, 218, 221
\seq_if_empty:NTF ....... 197
\seq_item:N ............. 113, 115, 122, 126, 133, 137, 171,
                        245, 301, 303, 310, 396, 397, 665, 666
\seq_log:N . 172, 178, 196, 207, 361, 376
\seq_map_break: ........ 409

156