The tagpdf-tree module
Commands trees and main dictionaries part of the tagpdf package

Ulrike Fischer[*]

Version 0.82, released 2021-06-14

```
⟨@@=tag⟩
⟨*tree⟩
\ProvidesExplPackage {tagpdf-tree-code} {2021-06-14} {0.82}
 {part of tagpdf - code related to writing trees and dictionaries to the pdf}
```

1 Trees 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

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

1.1 Structure tree

The following commands are needed to write out the structure.

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

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

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

The StructTreeRoot must be added to the catalog. TODO Should this be done later to prevent other packages to interfere?

```
\hook_gput_code:nnn{begindocument}{tagpdf}
  {
```

[*]E-mail: fischer@troubleshooting-tex.de

```
    \bool_if:NT \g__tag_active_struct_bool
      {
        \pdfmanagement_add:nnx
          { Catalog }
          { StructTreeRoot }
          { \pdf_object_ref:n { __tag/struct/0 } }
      }
  }
```

__tag_tree_write_structtreeroot: This writes out the root object.

```
\cs_new_protected:Npn \__tag_tree_write_structtreeroot:
  {
    \__tag_struct_write_obj:n { 0 }
  }
```

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

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

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

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

1.2 ParentTree

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

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

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

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

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

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

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

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

\g__tag_parenttree_objr_tl

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

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

__tag_parenttree_add_objr:nn

This command stores a Structparent number and a objref into the tl var. This only for objects like annotations, pages are handled elsewhere.

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

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

\l__tag_parenttree_content_tl

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

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

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

__tag_tree_fill_parenttree:

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

```
\cs_new_protected:Npn \__tag_tree_fill_parenttree:
  {
    \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{abspage}{-1}} %not quite clear i
      { %page ##1
        \prop_clear:N \l__tag_tmpa_prop
        \int_step_inline:nnnn{1}{1}{\__tag_ref_value_lastpage:nn{tagmcabs}{-1}}
          {
            %mcid####1
            \int_compare:nT
              {\__tag_ref_value:enn{mcid-####1}{tagabspage}{-1}=##1} %mcid is on current page
              {% yes
                \prop_put:Nxx
                  \l__tag_tmpa_prop
                  {\__tag_ref_value:enn{mcid-####1}{tagmcid}{-1}}
                  {\prop_item:Nn \g__tag_mc_parenttree_prop {####1}}
              }
          }
        \tl_put_right:Nx\l__tag_parenttree_content_tl
          {
            \int_eval:n {##1-1}\c_space_tl
            [\c_space_tl %]
          }
        \int_step_inline:nnnn
          {0}
          {1}
          { \prop_map_function:NN \l__tag_tmpa_prop\__tag_prop_count:nn -1 }
          {
            \prop_get:NnNTF \l__tag_tmpa_prop {####1} \l__tag_tmpa_tl
              {% page#1:mcid##1:\l__tag_tmpa_tl :content
                \tl_put_right:Nx \l__tag_parenttree_content_tl
```

```
                  {
                    \prop_item:cn { g__tag_struct_\l__tag_tmpa_tl _prop } {objref}
                    \c_space_tl
                  }
              }
              {
                \msg_warning:nn { tag } {tree-mcid-index-wrong}
              }
          }
        \tl_put_right:Nn
          \l__tag_parenttree_content_tl
          {%[
            ]^^J
          }
      }
  }
```

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

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

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

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

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

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

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

1.3 Rolemap dictionary

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

__tag/tree/rolemap

At first we reserve again an object.

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

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

__tag_tree_write_rolemap:

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

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

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

1.4 Classmap dictionary

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

__tag_tree_write_classmap:

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

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

1.5 Namespaces

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

`__tag/tree/namespaces`

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

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

`\__tag_tree_write_namespaces:`

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

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

1.6 Finishing the structure

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

`\__tag_finish_structure:`

```
\cs_new_protected:Npn \__tag_finish_structure:
  {
    \__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:`.)

1.7 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}}
         }
      }
  }
⟨/tree⟩
```

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.