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

Ulrike Fischer*

Version 0.82, released 2021-06-14

1 Public Commands

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

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

`\tag_struct_use:n` `\tag_struct_use:n{⟨label⟩}`

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

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

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

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

`\tag_struct_parent_int:` `\tag_struct_parent_int:`

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

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

2 Public keys

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

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

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

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

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

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

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

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

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

AF — AF = ⟨*object name*⟩
AFinline — AF-inline = ⟨*text content*⟩
AFinline-o

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.

```
⟨@@=tag⟩
⟨*struct⟩
\ProvidesExplPackage {tagpdf-struct-code} {2021-06-14} {0.82}
 {part of tagpdf - code related to storing structure}
```

2.1 Variables

\c@g__tag_struct_abs_int

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

```
\newcounter  { g__tag_struct_abs_int }
\int_gzero:N \c@g__tag_struct_abs_int
```

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

\g__tag_struct_objR_seq

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

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

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

\g__tag_struct_stack_seq

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

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

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

\g__tag_struct_tag_stack_seq

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

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

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

\g__tag_struct_stack_current_tl
\l__tag_struct_stack_parent_tmpa_tl

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

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

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

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

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

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

All properties have at least the keys

objnum the object number reference, TODO: check against xelatex/dvips

Type StructTreeRoot or StructElem

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

`\c__tag_struct_StructTreeRoot_entries_seq`
`\c__tag_struct_StructElem_entries_seq`

This contains the keys we support in the two object types. They need to be adapted if there are changes in the PDF format.

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

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

(*End definition for* `\c__tag_struct_StructTreeRoot_entries_seq` *and* `\c__tag_struct_StructElem_entries_seq`.)

2.1.1 Variables used by the keys

`\g__tag_struct_tag_tl`
`\g__tag_struct_tag_NS_tl`

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

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

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

`\l__tag_struct_key_label_tl`

This will hold the label value.

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

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

`\l__tag_struct_elem_stash_bool`

This will keep track of the stash status

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

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

2.2 Commands

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

`\__tag_struct_output_prop_aux:nn`
`\__tag_new_output_prop_handler:n`

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

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

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

2.2.1 Initialization of the StructTreeRoot

The first structure element, the StructTreeRoot is special, so created manually. The underlying object `@@/struct/0` and the other objects are setup in the tree code, which must be loaded first.

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

g__tag_struct_0_prop
g__tag_struct_kids_0_seq

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

\__tag_prop_gput:cnx
  { g__tag_struct_0_prop }
  { objref}
  { \pdf_object_ref:n { __tag/struct/0 } }

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


\__tag_prop_gput:cnx
  { g__tag_struct_0_prop }
  { ParentTree }
  { \pdf_object_ref:n { __tag/tree/parenttree } }

\__tag_prop_gput:cnx
  { g__tag_struct_0_prop }
  { RoleMap }
  { \pdf_object_ref:n { __tag/tree/rolemap } }

```

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

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

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

2.2.2 Handlings kids

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

__tag_struct_kid_mc_gput_right:nn

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

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

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

`\__tag_struct_kid_struct_gput_right:nn` `\__tag_struct_kid_struct_gput_right:xx`

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

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

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

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

`\__tag_struct_kid_OBJR_gput_right:nn` `\__tag_struct_kid_OBJR_gput_right:xx`

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

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

\cs_generate_variant:Nn\__tag_struct_kid_OBJR_gput_right:nn { xx }

```

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

__tag_struct_exchange_kid_command:N
__tag_struct_exchange_kid_command:c

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

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

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

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

__tag_struct_fill_kid_key:n

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

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

```
            [
              \seq_use:cn
                {
                  g__tag_struct_kids_#1_seq
                }
                {
                  \c_space_tl
                }
            ]
          }
      }
  }

```

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

`\__tag_struct_get_dict_content:nN` This maps the dictionary content of a structure into a tl-var. Basically it does what `\pdfdict_use:n` does. TODO!! this looks over-complicated. Check if it can be done with pdfdict now. Probably I need to get rid of non-object entries in the prop first. Don't forget that lua uses the objref entry in the lua code.

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

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

`\__tag_struct_write_obj:n` This writes out the structure object.

```
\cs_new_protected:Npn \__tag_struct_write_obj:n #1 % #1 is the struct num
  {
    \pdf_object_if_exist:nTF { __tag/struct/#1 }
      {
        \__tag_struct_fill_kid_key:n { #1 }
        \__tag_struct_get_dict_content:nN { #1 } \l__tag_tmpa_tl
        \exp_args:Nx
          \pdf_object_write:nx
```

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

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

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

Annotations used as structure content must

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

For a link this looks like this

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

```
\cs_new_protected:Npn \__tag_struct_insert_annot:nn #1 #2 %#1 object reference to the annotat
                                                        %#2 structparent number
  {
    \bool_if:NT \g__tag_active_struct_bool
      {
        %get the number of the parent structure:
        \seq_get:NNF
          \g__tag_struct_stack_seq
          \l__tag_struct_stack_parent_tmpa_tl
          {
            \msg_error:nn { tag } { struct-faulty-nesting }
          }
        %put the obj number of the annot in the kid entry, this also creates
        %the OBJR object
        \__tag_struct_kid_OBJR_gput_right:xx
          {
            \l__tag_struct_stack_parent_tmpa_tl
          }
          {
            #1 %
```

```
          }
        % add the parent obj number to the parent tree:
        \exp_args:Nnx
        \__tag_parenttree_add_objr:nn
          {
            #2
          }
          {
            \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
          }
        % increase the int:
        \stepcounter{ g__tag_parenttree_obj_int }
      }
  }
```

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

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

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

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

2.3 Keys

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

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

```
\cs_generate_variant:Nn \seq_set_split:Nnn{Nne}
\keys_define:nn { __tag / struct }
  {
    label .tl_set:N      = \l__tag_struct_key_label_tl,
    stash .bool_set:N    = \l__tag_struct_elem_stash_bool,
    tag   .code:n        = % S property
      {
        \seq_set_split:Nne \l__tag_tmpa_seq { / } {#1/\prop_item:Nn\g__tag_role_tags_NS_prop{
        \tl_gset:Nx \g__tag_struct_tag_tl    { \seq_item:Nn\l__tag_tmpa_seq {1} }
        \tl_gset:Nx \g__tag_struct_tag_NS_tl { \seq_item:Nn\l__tag_tmpa_seq {2} }
        \bool_if:NT \g__tag_check_tags_bool
          {
            \__tag_check_structure_tag:N \g__tag_struct_tag_tl
          }
       \__tag_prop_gput:cnx
         { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
         { S }
         { \pdf_name_from_unicode_e:n{ \g__tag_struct_tag_tl} } %
```

```
      \prop_get:NVNT \g__tag_role_NS_prop\g__tag_struct_tag_NS_tl\l__tag_tmpa_tl
        {
          \__tag_prop_gput:cnx
           { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
           { NS }
           { \l__tag_tmpa_tl } %
        }
    },
  title .code:n        = % T property
    {
      \str_set_convert:Nnon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { T }
        { <\l__tag_tmpa_str> }
    },
  title-o .code:n        = % T property
    {
      \str_set_convert:Nnon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { T }
        { <\l__tag_tmpa_str> }
    },
  alttext .code:n      = % Alt property
    {
      \str_set_convert:Nnon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { Alt }
        { <\l__tag_tmpa_str> }
    },
  alttext-o .code:n      = % Alt property
    {
      \str_set_convert:Noon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { Alt }
```

```
        { <\l__tag_tmpa_str> }
    },
  actualtext .code:n  = % ActualText property
    {
      \str_set_convert:Nnon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { ActualText }
        { <\l__tag_tmpa_str>}
    },
  actualtext-o .code:n  = % ActualText property
    {
      \str_set_convert:Noon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { ActualText }
        { <\l__tag_tmpa_str>}
    },
  lang .code:n        = % Lang property
    {
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { Lang }
        { (#1) }
    },
  ref .code:n        = % Lang property
    {
      \tl_clear:N\l__tag_tmpa_tl
      \clist_map_inline:nn {#1}
        {
          \tl_put_right:Nx \l__tag_tmpa_tl
            {~\ref_value:nn{tagpdfstruct-##1}{tagstructobj} }
        }
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
        { Ref }
        { [\l__tag_tmpa_tl] }
    },
  E .code:n        = % E property
    {
      \str_set_convert:Nnon
        \l__tag_tmpa_str
        { #1 }
        { default }
        { utf16/hex }
      \__tag_prop_gput:cnx
```

```
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { E }
          { <\l__tag_tmpa_str> }
      },
  }
```

(*End definition for* `label` *and others. These functions are documented on page* 2.)

AF
AFinline
AFinline-o

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

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

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

```
      \__tag_prop_gput:cnx
        { g__tag_struct_\int_use:N\c@g__tag_struct_abs_int _prop }
        { AF }
        { \pdf_object_ref:e {__tag/fileobj\int_use:N\c@g__tag_struct_abs_int } }
      \group_end:
    }
}
```

(*End definition for* `AF`, `AFinline`, *and* `AFinline-o`. *These functions are documented on page 3.*)

2.4 User commands

```
\cs_generate_variant:Nn \pdf_object_ref:n {e} % check later
```

`\tag_struct_begin:n`
`\tag_struct_end:`

```
\cs_new_protected:Npn \tag_struct_begin:n #1 %#1 key-val
 {
   \group_begin:
   \int_gincr:N \c@g__tag_struct_abs_int
   \__tag_prop_new:c  { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
   \__tag_new_output_prop_handler:n {\int_eval:n { \c@g__tag_struct_abs_int }}
   \__tag_seq_new:c  { g__tag_struct_kids_\int_eval:n { \c@g__tag_struct_abs_int }_seq}
   \exp_args:Ne
     \pdf_object_new:nn
       { __tag/struct/\int_eval:n { \c@g__tag_struct_abs_int } }
       { dict }
   \__tag_prop_gput:cnx
     { g__tag_struct_\int_eval:n { \c@g__tag_struct_abs_int }_prop }
     { objref}
     {
       \exp_args:Ne
         \pdf_object_ref: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 }
   \keys_set:nn { __tag / struct} { #1 }
   \__tag_check_structure_has_tag:n { \int_eval:n {\c@g__tag_struct_abs_int} }
   \tl_if_empty:NF
     \l__tag_struct_key_label_tl
     {
       \__tag_ref_label:en{tagpdfstruct-\l__tag_struct_key_label_tl}{struct}
     }
   %get the potential parent from the stack:
   \seq_get:NNF
     \g__tag_struct_stack_seq
     \l__tag_struct_stack_parent_tmpa_tl
     {
       \msg_error:nn { tag } { struct-faulty-nesting }
     }
   \seq_gpush:NV \g__tag_struct_stack_seq        \c@g__tag_struct_abs_int
   \seq_gpush:NV \g__tag_struct_tag_stack_seq    \g__tag_struct_tag_tl
   \tl_gset:NV   \g__tag_struct_stack_current_tl \c@g__tag_struct_abs_int
```

```
    %\seq_show:N   \g__tag_struct_stack_seq
    \bool_if:NF
      \l__tag_struct_elem_stash_bool
      {%set the  parent
        \__tag_prop_gput:cnx
          { g__tag_struct_\int_eval:n {\c@g__tag_struct_abs_int}_prop }
          { P }
          {
            \pdf_object_ref:e { __tag/struct/\l__tag_struct_stack_parent_tmpa_tl }
          }
        %record this structure as kid:
        %\tl_show:N \g__tag_struct_stack_current_tl
        %\tl_show:N \l__tag_struct_stack_parent_tmpa_tl
        \__tag_struct_kid_struct_gput_right:xx
           { \l__tag_struct_stack_parent_tmpa_tl }
           { \g__tag_struct_stack_current_tl }
        %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
        %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
      }
    %\prop_show:c { g__tag_struct_\g__tag_struct_stack_current_tl _prop }
    %\seq_show:c {g__tag_struct_kids_\l__tag_struct_stack_parent_tmpa_tl _seq}
    \group_end:
  }


\cs_new_protected:Nn \tag_struct_end:
  { %take the current structure num from the stack:
    %the objects are written later, lua mode hasn't all needed info yet
    %\seq_show:N \g__tag_struct_stack_seq
    \seq_gpop:NN   \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
    \seq_gpop:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
      {
        \int_compare:nNnT {\l__tag_loglevel_int} > { 0 }
          {
            \__tag_check_info_closing_struct:o { \g__tag_struct_stack_current_tl }
          }
      }
      { \__tag_check_no_open_struct: }
    % get the previous one, shouldn't be empty as the root should be there
    \seq_get:NNTF \g__tag_struct_stack_seq \l__tag_tmpa_tl
      {
        \tl_gset:NV   \g__tag_struct_stack_current_tl \l__tag_tmpa_tl
      }
      {
        \__tag_check_no_open_struct:
      }
   \seq_get:NNT \g__tag_struct_tag_stack_seq \l__tag_tmpa_tl
      {
        \tl_gset:NV \g__tag_struct_tag_tl \l__tag_tmpa_tl
      }
  }
```

(*End definition for* `\tag_struct_begin:n` *and* `\tag_struct_end:`*. These functions are documented on page 1.*)

\tag_struct_use:n This command allows to use a stashed structure in another place.

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

(*End definition for* `\tag_struct_use:n`*. This function is documented on page 1.*)

\tag_struct_insert_annot:nn
\tag_struct_insert_annot:xx
\tag_struct_parent_int:

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

It must be used together with

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

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

⟨/struct⟩

```

(*End definition for* `\tag_struct_insert_annot:nn` *and* `\tag_struct_parent_int:`*. These functions are documented on page 1.*)

```
⟨*attr⟩
\ProvidesExplPackage {tagpdf-attr-code} {2021-06-14} {0.82}
  {part of tagpdf - code related to attributes and attribute classes}

% the obj is written in tagpdf-tree-code.

\seq_new:N  \g__tag_attr_class_used_seq
\prop_new:N \g__tag_attr_objref_prop %will contain obj num of used attributes

\prop_new:N \g__tag_attr_entries_prop
```

```
\tl_new:N   \g__tag_attr_class_content_tl
\tl_new:N   \l__tag_attr_objtmp_tl
\tl_new:N   \l__tag_attr_value_tl


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

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


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

```
  }

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