The l3pdfmanagement module
Managing central PDF resources
LaTeX PDF management bundle

The LaTeX Project*

Version 0.96z, released 2026-04-15

1 l3pdfmanagement documentation

When creating a pdf a number of objects, dictionaries and entries to central "core" dictionaries must be created.

The commands in this module offer interfaces to this core PDF dictionaries They unify a number of primitives like the pdftex registers and commands `\pdfcatalog`, `\pdfpageattr`, `\pdfpagesattr`, `\pdfinfo`, `\pdfpageresources` and similar commands of the other backends in a backend independent way.

The supported backends are pdflatex, lualatex, (x)dvipdfmx (latex, xelatex and—starting in texlive 2021–lualatex) and dvips with ps2pdf (not completely yet). dvips with distiller could work too but is untested.

That the interfaces are backend independent doesn't mean that the results and even the compilation behavior is identical. The backends are too different to allow this. Some backends expand arguments e.g. in a `\special` while other don't. Some backends can insert a resource at the first compilation, while another uses the aux-file and a label and so needs at least two. Some backends create and manage resources automatically which must be managed manually by other backends.

The dictionaries and resources handled by this module are inserted only once in a PDF or only once per page. Examples are the Catalog dictionary, the Info dictionary, the page resources. For these dictionaries and resources management by the LaTeX kernel is necessary to avoid that packages overwrite settings from other packages which would lead to clashes and incompatibilities. It is therefore necessary that *all* packages which want to add content to these dictionaries and resources use the interface provided by this module.

As these dictionaries and resources are so central for the PDF format values to these dictionaries are always added globally. Through the interface values can be added (and in many cases also removed) by users and packages, but the actually writing of the dictionary entries and resources to the PDF is handled by the kernel code.

The interface uses as main name to address the resources *Paths* which follow the names and structure described in the PDF reference. This should make it easy to identify

*E-mail: latex-team@latex-project.org

the names needed to insert a specific PDF resources with the new interfaces. All *Paths* have names starting with an uppercase letter.

The following tabular summarize the *Paths* and which pdftex primitive they replace:

Info	`\pdfinfo`
Catalog & various subdictionaries	`\pdfcatalog`
Pages	`\pdfpagesattr`
Page, ThisPage	`\pdfpageattr`
Page/Resources/ExtGState	`\pdfpageresources`
Page/Resources/Shading	`\pdfpageresources`
Page/Resources/Pattern	`\pdfpageresources`
Page/Resources/ColorSpace	`\pdfpageresources`

There is no `Page/Resources/Properties` dictionary in the list, because this dictionary is not filled directly, but managed through side effects when setting BDC-marks.

1.1 User Commands

To avoid problems with older documents the resource management of this module is not activated unconditionally. The values are pushed out to the dictionaries only if a boolean has been set to true. The state can be tested with a conditional.

`\pdfmanagement_if_active_p:` ⋆
`\pdfmanagement_if_active:`*TF* ⋆

New: 2020-07-04

This conditional tests if the resource management code is active.

`\IfPDFManagementActiveTF` This is a LaTeX2e version of the conditional

New: 2021-07-23

\pdfmanagement_add:nnn
\pdfmanagement_add:(nne|nee|eee)
\PDFManagementAdd

New: 2020-04-06
Updated: 2021-07-23

\pdfmanagement_add:nnn {⟨*resource path*⟩} {⟨*name*⟩} {⟨*value*⟩}

This function puts {⟨*name*⟩} {⟨*value*⟩} in the PDF resource described by the symbolic name {⟨*resource path*⟩}. Technically it stores it globally in an internal property lists and writes it later into the right PDF dictionary[1] Which values for {⟨*resource path*⟩} exist is described in the following. {⟨*name*⟩} should be a PDF Name without the starting slash. Like with all keys used in PDF dictionaries (see the l3pdfdict module) the name is escaped with \str_convert_pdfname:n when stored. {⟨*value*⟩} should be a valid PDF value for this Name in the target dictionary. \PDFManagementAdd is a copy of \pdfmanagement_add:eee and so expands all its arguments.

The code works with all major engines but not necessarily in the same way. Most importantly

- The expansion behaviour of the backends can differ. Some backends expand a value always fully when writing to the PDF, with other backends command names could end as strings in the PDF. So one should neither rely on {⟨*name*⟩} {⟨*value*⟩} to be expanded nor not expanded by the backend commands.
- The number of compilations needed can differ between the engines and backends. Some engines have to use labels and the aux-file to setup the dictionaries and so need at least two compilations to put everything in place.
- dvips doesn't support everything. It is for example not possible to add manually or through side effects a name tree like /AP or /JavaScript, pdfmark doesn't provide a handler here—at least I didn't find anything suitable.

\pdfmanagement_show:n

New: 2020-04-08

\pdfmanagement_show:n {⟨*resource path*⟩}

This shows the content of the dictionary targeted by {⟨*resource path*⟩} in the log and on the terminal if possible.

It is not reliable for page resources as these are filled at shipout.

It also doesn't show necessarily all the content. For example most backends add automatically entries to the Info dictionary.

\pdfmanagement_remove:nn

New: 2020-04-07

\pdfmanagement_remove:nn {⟨*resource path*⟩} {⟨*name*⟩}

Removes /⟨*name*⟩ and its associated ⟨*value*⟩ from the dictionary described with {⟨*resource path*⟩} The removal is global. If ⟨*name*⟩ is not found no change occurs, *i.e* there is no need to test for the existence of a name before trying to remove it. Values from the special Catalog entries where the values are collected in arrays can't be removed (but should ever a use case appear it could be added).

[1] Currently all resources are PDF dictionaries, so resource and dictionary mean the same.

1.2 Description of the resource paths

1.2.1 Info: The Info dictionary

If the primitive commands of the engines are used too there will be double entries in the pdf (at least with the backend pdftex and luatex). How pdf viewer handles this is unpredictable.

pdfmanagement: Info `\pdfmanagement_add:nnn {Info} {⟨name⟩} {⟨value⟩}`

Adds `/⟨name⟩` and the ⟨*value*⟩ to the Info dictionary. ⟨*name*⟩ should be a PDF name without the leading slash, Like with all keys used in PDF dictionaries (see the l3pdfdict module) the name is escaped with `\str_convert_pdfname:n` when stored. ⟨*value*⟩ should be a valid pdf value. Any escaping or (re)encoding must be done explicitly. If a ⟨*name*⟩ is used twice, only the last ⟨*value*⟩ set will be used. The Info dictionary is written at the end of the compilation, so values can be set at any time. The Info dictionary expects utf16be in the strings, so a conversion like this is normally sensible:

```
\str_set_convert:Nnnn \l_tmpa_str { Grüße }{ default } {utf16/string}
\pdfmanagement_add:nne {Info} {Title}{(\l_tmpa_str)}
```

The entries in Info dictionary are rather special as the engines/backends adds some core entries, and changing or removing these entries is not always possible.

The special entries are

Producer Added by all engines and backends. Removing the entry is only possible with luatex with `\pdfvariable suppressoptionalinfo 128`. Changing is possible with `\pdfmanagement_add:nnn` with the exception of dvips/pstopdf where the entry is always something like `GPL Ghostscript 9.53.3`.

Creator Added by all engines and backends. Removal only possible in luatex by adding 16 to the bitset. Changing is possible with the management command.

CreationDate Added by all engines and backends. With the exception of `dvips/ps2pdf` `SOURCE_DATE_EPOCH` is honored. With pdftex it is possible to suppress it with `\pdfinfoomitdate = 1`, and in luatex by adding 32 to the bitset. Changing is possible with the management command and will overwrite an epoch setting.

ModDate Added by all engines and backends with the exception of xdvipdfmx. With the exception of `dvips/ps2pdf SOURCE_DATE_EPOCH` is honored. Suppressing it is possible in pdftex with `\pdfinfoomitdate = 1`, and in luatex by adding 64 to the bitset. Changing is possible with the management command.

Trapped Added by pdftex and luatex. Removal only possible in luatex by adding 256 to the bitset. Changing (and adding in the other backends) is possible with the management command.

PTEX.Fullbanner Added by pdftex and luatex. Removal possible in pdftex with `\pdfsuppressptexinfo-1`, in luatex by adding 512 to the bitset. Changing is not possible.

Title Added by dvips/ps2pdf and set to `filename.dvi`. Removal is probably not possible, but it can be overwritten with the management command.

1.2.2 Pages: The "Pages" dictionary

As the content of this dictionary is written at the end it will in pdftex and luatex overwrite values added with the primitive commands (e.g. `\pdfpagesattr`. Package authors should use the management commands instead.

By using this path with the pdfmanagement interface, values can be added to the /Pages object. This replaces for example `\pdfpagesattr`.

pdfmanagement: Pages `\pdfmanagement_add:nnn {Pages} {⟨name⟩} {⟨value⟩}`

Adds `/⟨name⟩ ⟨value⟩` to the `/Pages` dictionary. It is always stored globally. The content is written to the pdf at the end of the compilation, so values can be added, changed or removed until then. ⟨*name*⟩ should be a valid pdf name without the leading slash, ⟨*value*⟩ should be a valid pdf value. Any escaping or (re)encoding must be done explicitly. Some backends expand the value but this should not be relied on. If a ⟨*name*⟩ is used twice, only the last ⟨*value*⟩ set will be used.

1.2.3 "Page" and "ThisPage"

pdfmanagement: Page `\pdfmanagement_add:nnn {Page} {⟨name⟩} {⟨value⟩}`

New: 2020-04-12

Values added with the path `Page` are added to the page dictionary of the current page and the following pages. The current page means the page on which the command is *executed*. ⟨*name*⟩ should be a valid pdf name without the leading slash. Typical names used here are e.g. `Rotate` and `CropBox`. ⟨*value*⟩ should be a valid pdf value. Any escaping or (re)encoding must be done explicitly. Some backends expand the value but this should not be relied on. To avoid problems with the asynchronous page breaking the command should be used after `\newpage` or in the header. It should not be used in a float, as it will then quite probably be executed on the wrong page. The value is assigned directly and is always stored globally. If a ⟨*name*⟩ is used twice, only the last ⟨*value*⟩ set will be used. Names set with `\pdfmanagement_add:nnn{ThisPage}` will overwrite names set with `\pdfmanagement_add:nnn{Page}` if there is a clash. Values can be removed again with `\pdfmanagement_remove:nn`. This replaces `\pdfpageattr`.

pdfmanagement: ThisPage `\pdfmanagement_add:nnn {ThisPage} {⟨name⟩} {⟨value⟩}`

New: 2020-04-12

Adds `/⟨name⟩ ⟨value⟩` at *shipout* to the page dictionary of the current page. Current page means here the *shipout* page. *shipout* means at the end of the `shipout/background` hook. Code that wants to set a value in a shipout hook should use the `shipout/background` hook too. Other hooks are either too early or too late. It is always stored globally. If `{⟨name⟩}` has already a value set in the `Page` dictionary it will be overwritten for this page. ⟨*name*⟩ should be a valid pdf name without the leading slash, ⟨*value*⟩ should be a valid pdf value. Any escaping or (re)encoding must be done explicitly. If a ⟨*name*⟩ is used twice, only the last ⟨*value*⟩ set will be used. With the engine pdflatex (at least) a second compilation is needed. Values added to `ThisPage` can not be removed. It is not possible to show the content of this dictionary with `\pdfmanagement_show:n`.

Changing the `/MediaBox` : It is possible to change the `/MediaBox` of one or more pages by setting it for the `Page` or `ThisPage` path (using `Pages` doesn't work, the engines overwrite this)—this works even with dvips and allows to create pages of different sizes. But you must be careful with the values. If you set e.g. with pdflatex `\pdfpageheight`

to 300bp you get a mediabox of `0 0 595 300`, but pdflatex measure from the top and will also move the reference point up, so effectively you get the *upper* third of the page. If you set the `/MediaBox` to `0 0 595 300` with `\pdfmanagement_add:nnn` you get the *lower* third. In general it is better to use only the primitive commands to avoid confusing results.

1.2.4 "Page/Resources": ExtGState, ColorSpace, Shading, Pattern

`pdfmanagement: Page/Resources/ExtGState`
`pdfmanagement: Page/Resources/ColorSpace`
`pdfmanagement: Page/Resources/Shading`
`pdfmanagement: Page/Resources/Pattern`

`\pdfmanagement_add:nnn {Page/Resources/⟨resource⟩} {⟨name⟩} {⟨value⟩}`

Updated: 2020-04-10

Adds `/⟨name⟩ ⟨value⟩` to the page resource ⟨*resource*⟩. ⟨*resource*⟩ can be `ExtGState`, `ColorSpace`, `Pattern` or `Shading`. The values are always stored globally. The content is written to the pdf at the end of the compilation, so values can be added until then. ⟨*name*⟩ should be a valid pdf name without the leading slash, ⟨*value*⟩ should be a valid pdf value for the resource. Any escaping or (re)encoding must be done explicitly. If a ⟨*name*⟩ is used twice, only the last ⟨*value*⟩ set will be used.

With the dvips backend the command does nothing: these resources are managed by ghostscript or the distiller if e. g. transparency is used.

The resources are added to all pages starting with the first where something has been added to a resources. That means that for example all ExtGState resources are combined in one dictionary object and every page with a ExtGState resource refer to this object [2].

The primitive commands (e.g. `\pdfpageresources`) to set the resources should not be used together with this code as the calls will overwrite each other and values will be lost. This means that currently there are clashes with the packages tikz, transparent and colorspace.

1.2.5 "Catalog" & subdirectories

The catalog is a central dictionary in a PDF with a number of subdictionaries. Entries to the top level of the catalog can be added with `\pdfmanagement_add:nnn {Catalog}{⟨Name⟩}{⟨Value⟩}`. Entries to subdictionaries by using in the first argument one of the paths described later. The entries in the catalog have varying requirements regarding the PDF management. Some entries (like `/Lang`) are simple values where new values should overwrite existing values, other like for example `/OutputIntents` can contain a number of values and can be filled from more than one source. In some cases the values that needs to be added are not at the top-level but in some subsubdictionary or are actually part of an array. To handle the pdf management uses a variety of internal, special handlers.

In some cases entries are added implicitly. For example entries to the name tree of the `/EmbeddedFiles` key in the `/Names` directory are added with the commands of the `l3pdffile` module. This clashes with e.g. the embedfile package which should not be used!

[2] This is similar to how pgf handles this resources

Entries at the top level of the catalog The Names in the following tabular are entries that are added to the top level of the catalog.

If ⟨*Name*⟩ gets assigned a value more than once the last one wins. There is no check that the values have the correct type and format. It is up to the user to ensure that the value does what is intended.

The required PDF version is only mentioned if it is larger than 1.5.

Example: `\pdfmanagement_add:nnn {Catalog}{PageMode}{/UseNone}`

Name	**Value**	**Remark**
Collection	objref or dict	the content should be build by external packages (see eg embedfile)
DPartRoot	objref or dict	PDF 2.0
Lang	string	e.g. `(de-DE)`
Legal	objref or dict	
Metadata	objref or stream	
NeedsRendering	boolean	PDF 1.7
OpenAction	array (dest) or dict (action)	
PageLabels	objref or dict	number tree
PageLayout	name	one of /SinglePage, /OneColumn, /TwoColumnLeft, /TwoColumnRight, /TwoPageLeft, /TwoPageRight
PageMode	name	one of /UseNone, /UseOutlines, /UseThumbs, /UseOC, /UseAttachments (PDF 1.6)
Perms	objref or dict	permissions
PieceInfo	objref or dict	
SpiderInfo	objref or dict	
StructTreeRoot	objref or dict	
Threads	objref to an array	
URI	objref or dict	
Version	name	eg. `/1.7`
⟨*unknown*⟩		an unknown ⟨*name*⟩ will be inserted without a warning.

Simple entries in subdictionaries of the catalog The following resource paths have been predeclared and allow to add values to the respective subdictionaries of the catalog. The names of the dictionaries follow the naming and location of the dictionaries in the PDF reference. If ⟨*Name*⟩ gets assigned two values the last one wins.

Example: `\pdfmanagement_add:nnn {Catalog/MarkInfo}{Marked}{true}`

Path/dictionary	Names	Value	Remark
Catalog/AA	WC, WS, DS, WP,DP	all dict	
Catalog/AcroForm	NeedAppearances	boolean	In pdf 2.0 NeedAppearances is deprecated, it is then required that every widget has an appearance streams.
	SigFlags	Integer	
	DA	String	
	Q	Integer	
	XFA	stream or array	pdf 1.5
Catalog/AcroForm/DR	⟨*name*⟩		probably unneeded
Catalog/AcroForm/DR/Font	⟨*name*⟩	dict	
Catalog/MarkInfo	Marked	boolean	
	UserProperties	boolean	
	Suspects	boolean	
Catalog/ViewerPreferences	HideToolbar	boolean	
	Direction	/R2L or /L2R	
	…		many more, see the reference

Catalog entries with multiple values in arrays The following entries are special: Their values are arrays and it must be possible to append to such arrays. This means that a new call to set this value doesn't replace the value but appends it. The value is an object reference. It is sensible to declare the object first. E.g.

```
\pdf_object_new:n     {module/intent}
\pdf_object_write:nnn {module/intent}{dict}{...}
\pdfmanagement_add:nne  {Catalog} {OutputIntents}{\pdf_object_ref:n {module/intent}}
```

or

```
\pdf_object_unnamed_write:nn   {dict} { ... }
\pdfmanagement_add:nne  {Catalog} {OutputIntents}{\pdf_object_ref_last:}
```

Path/dictionary	Name	Value	Remark
Catalog/AcroForm	Fields	object reference	
Catalog/AcroForm	CO	object reference	
Catalog	AF	object reference	PDF 2.0, associated files
Catalog/OCProperties	OCGs	object reference	if there are OCProperties, OCGs and D are required.
Catalog/OCProperties	Configs	object reference	
Catalog/OCProperties	D	object reference	This is actually a single value as there can be only one default. If the value is set twice, the second wins, and the first is added to OCProperties/Configs.
Catalog	OutputIntents	object reference	
Catalog	Requirements	object reference	PDF 1.7
Catalog/Names	EmbeddedFiles	object reference	This should reference a filespec dictionary. It will attach the file to the file panel.

Catalog entries for name trees *Not supported in the dvips backend, pdfmark doesn't have an interface here.*

In various places the PDF format allows to reference objects by name instead of by object reference. The relationship between a name and the object reference are store in so-called *name trees*, which are stored in the Catalog/Names dictionary. The `/Dests` and the `/EmbeddedFiles` name trees are handled implicitly if destinations or files are added. Names to the other name trees can be added with `\pdfmanagement_add:nnn`, e.g. to add an value to the AP names (for appearance streams) use

```
\pdfmanagement_add:nne  { Catalog / Names / AP } {myAPname} {\pdf_object_ref_last:}
```

Remarks:

- The name `myAPname` is processed through `\pdf_string_from_unicode:nnN{utf8/string}` and parentheses are added automatically. Ensure that the use of the name handles it in the same way.
- It is currently not possible to test if a name has already been used by another package or previous code, so use names where you can be confident that they are unique. (It would be possible to split up the first part and test, but it would slow down the compilation and I'm not sure if it is worth the trouble)
- The value is not preprocessed, it is up-to-you to ensure that it does the right thing.
- Currently the structure of the name tree is flat, it doesn't use Kids. But this can be changed if the need arise.

The following name trees can be filled with this method. Currently only the first three are activated. For the first, `EmbeddedFiles` there are two methods to add a value: `\pdfmanagement_add:nnn{Catalog/Names/EmbeddedFiles}{name}{reference}` and `\pdfmanagement_` This is intended, the second methods creates a name on the fly (with the prefix l3ef)

Catalog/Names/EmbeddedFiles	A name tree mapping name strings to file specifications for en
Catalog/Names/AP	A name tree mapping name strings to annotation appearance
Catalog/Names/JavaScript	A name tree mapping name strings to documentlevel ECMAS
(inactive) Catalog/Names/Pages	A name tree mapping name strings to visible pages for use in
(inactive) Catalog/Names/Templates	A name tree mapping name strings to invisible pages for use
(inactive) Catalog/Names/IDS	A name tree mapping digital identifiers to Web.Capture conte
(inactive) Catalog/Names/URLS	A name tree mapping name strings to documentlevel ECMAS
(inactive) Catalog/Names/Renditions	A name tree mapping name strings (which shall have Unicode

2 l3pdfmanagement implementation

```
⟨@@=pdfmanagement⟩
⟨*header⟩
%
\ProvidesExplPackage{l3pdfmanagement}{2026-04-15}{0.96z}
  {Management of core PDF dictionaries (LaTeX PDF management bundle)}
⟨/header⟩
```

2.1 Messages

```
⟨*package⟩
\msg_new:nnn  { pdfmanagement } { unknown-dict }
              { The~PDF~management~resource~'#1'~is~unknown. }

\msg_new:nnn  { pdfmanagement } { empty-value }
              { The~value~for~#1~is~empty~and~will~be~ignored }

\msg_new:nnn  { pdfmanagement } { no-removal }
              { It~is~not~possible~to~remove~values~from~'#1'.}

\msg_new:nnn  { pdfmanagement } { no-show }
              { It~is~not~possible~to~show~the~content~of~'#1'.}

\msg_new:nnn  { pdfmanagement } { name-exist }
              { The~name~'#1'~has~already~been~used~for~name~tree~'#2'.}

\msg_new:nnn  { pdfmanagement } { show-dict }
  {
    The~PDF~resource~'#1'~
    \tl_if_empty:nTF {#2}
      { is~empty \\>~ . }
      { contains~the~pairs~(without~outer~braces): #2 . }
  }
\msg_new:nnn  { pdfmanagement } { dict-already-defined  }
  {
    The~path~'#1'~is~already~defined.
  }
\msg_new:nnn  { pdfmanagement } { inactive  }
```

```
  {
    The~PDF~resources~management~is~not~active\\
    command~'#1'~ignored.
  }
```

\l__pdfmanagement_tmpa_tl
\l__pdfmanagement_tmpb_tl
\l__pdfmanagement_tmpa_seq

Some temp variables

```
\tl_new:N \l__pdfmanagement_tmpa_tl
\tl_new:N \l__pdfmanagement_tmpb_tl
\seq_new:N \l__pdfmanagement_tmpa_seq
```

(*End of definition for* `\l__pdfmanagement_tmpa_tl`, `\l__pdfmanagement_tmpb_tl`, *and* `\l__pdfmanagement_tmpa_seq`.)

\g__pdfmanagement_active_bool

This boolean will controlled the activation of the management code. It is now a noop and always true.

```
\bool_new:N \g__pdfmanagement_active_bool
\bool_gset_true:N \g__pdfmanagement_active_bool
```

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

A user predicate to test if the management code is active

```
\prg_new_conditional:Npnn \__pdfmanagement_if_active:  { p , T , F , TF }
  {
    \prg_return_true:
  }
\prg_set_eq_conditional:NNn
  \pdfmanagement_if_active: \__pdfmanagement_if_active: { p , T , F , TF }

\cs_set_eq:NN \IfPDFManagementActiveTF\use_i:nn
```

We use a hook, to collect value added before the backend is ready.

```
\hook_new:n {pdfmanagement/add}
\cs_new_protected:Npn \pdfmanagement_add:nnn #1 #2 #3
  {
     \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
       {
         \hook_gput_code:nnn
           {pdfmanagement/add}
           {pdfmanagement}
           {
             \__pdfmanagement_handler_gput:nnn { #1 }{ #2 }{ #3 }
           }
       }
       {
         \msg_error:nnn{pdfmanagement}{unknown-dict}{#1}
       }
  }

\cs_generate_variant:Nn \pdfmanagement_add:nnn {nne,nee,eee,nnx,nxx,xxx}
\cs_set_eq:NN \PDFManagementAdd \pdfmanagement_add:eee
```

2.2 Hooks – shipout and end of run code

Code is executed in three places: At shipout of every page, at shipout of the last page, at the end of the document (after the last clearpage). Due to backend differences the

code in the three places (and the exact timing) can be different: pdflatex/lualatex can execute code after the last `\clearpage` which the dvi-based drivers have to add on a shipout page.

\g__kernel_pdfmanagement_end_run_code_tl This variables contain the code run in the three places.

```
\tl_new:N \g__kernel_pdfmanagement_thispage_shipout_code_tl
\tl_new:N \g__kernel_pdfmanagement_lastpage_shipout_code_tl
\tl_new:N \g__kernel_pdfmanagement_end_run_code_tl
```

(*End of definition for* `\g__kernel_pdfmanagement_thispage_shipout_code_tl` *and* `\g__kernel_pdfmanagement_lastpage_shipout_code_tl` *and* `\g__kernel_pdfmanagement_end_run_code_tl`.)

```
\tl_gset:Nn \g__kernel_pdfmanagement_thispage_shipout_code_tl
  {
    \exp_args:NV \__pdf_backend_ThisPage_gpush:n      { \g_shipout_readonly_int }
    \exp_args:NV \__pdf_backend_PageResources_gpush:n { \g_shipout_readonly_int }
  }

\tl_gset:Nn \g__kernel_pdfmanagement_end_run_code_tl
  {
    \__pdf_backend_PageResources_obj_gpush:          %ExtGState etc
    \__pdfmanagement_Pages_gpush:            %pagesattr
    \__pdfmanagement_Info_gpush:             %pdfinfo
    \__pdfmanagement_Catalog_gpush:
  }
```

2.3 Naming convention

Currently the following names are used: All have internally additionally a `Core` before the slash, to hide the real name a bit.

```
/Info                       %    (\pdfinfo)
/Catalog                    %    (\pdfcatalog)
/Catalog/AA                 %
/Catalog/AcroForm
/Catalog/OCProperties
/Catalog/OutputIntents
/Catalog/AcroForm/DR
/Catalog/AcroForm/DR/Font
/Catalog/MarkInfo
/Catalog/ViewerPreferences
/Pages                      %    (\pagesattr)
/Page                       %    (\pageattr)
/ThisPage                   %    (\pageattr)
/backend_PageN/Resources/Properties % this is only internal.
/Page/Resources/ExtGState
/Page/Resources/ColorSpace
/Page/Resources/Pattern
/Page/Resources/Shading
/Page/Resources/Properties
/Xform/Resources/Properties
```

__pdfmanagement_handler_gput:nnn
__pdfmanagement_get:nnN
__pdfmanagement_gremove:nn
__pdfmanagement_show:n

`\__pdfmanagement_handler_gput:nnn` is the main command to fill the dictionaries. In simple cases it directly fill the property list, but if a handler exists this is called. It is important to use it only in places where this make sense.

```
%global
\cs_new_protected:Npn \__pdfmanagement_handler_gput:nnn #1 #2 #3  %#1 dict, #2 name, #3 value
  {
    \tl_if_empty:nTF { #3 }
      {
        \msg_none:nnn { pdfmanagement }{ empty-value }{ /#1/#2 }
      }
      {
        \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
          {
            \cs_if_exist:cTF
              { __pdfmanagement_handler/#1/?_gput:nn } %general, name independent handler
              { \use:c {__pdfmanagement_handler/#1/?_gput:nn} {#2} {#3} }
              {
                \cs_if_exist:cTF
                  { __pdfmanagement_handler/#1/#2_gput:n }
                  { \use:c {__pdfmanagement_handler/#1/#2_gput:n} {#3} } %special handler
                  {
                    \exp_args:Nne
                    \prop_gput:cnn
                      { \__kernel_pdfdict_name:n { g__pdf_Core/#1 } }
                      { \str_convert_pdfname:n { #2 } }
                      { #3 }
                  }
              }
          }
          {
            \msg_error:nnn { pdfmanagement } { unknown-dict } { #1 }
          }
      }
  }


\cs_generate_variant:Nn \__pdfmanagement_handler_gput:nnn {nee}

\cs_new_protected:Npn \__pdfmanagement_get:nnN  #1 #2 #3 %path,key,macro
  {
    \exp_args:Nne
    \prop_get:cnN
      { \__kernel_pdfdict_name:n { g__pdf_Core/#1 } }
      { \str_convert_pdfname:n {#2} } #3
  }


\cs_new_protected:Npn \__pdfmanagement_handler_gremove:nn #1 #2 %path,key
  {
    \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
          {
            \cs_if_exist:cTF
              { __pdfmanagement_handler/#1/?_gremove:n } %general, name independent handler
```

```
              { \use:c {__pdfmanagement_handler/#1/?_gremove:n} {#2} }
              {
                \cs_if_exist:cTF
                  { __pdfmanagement_handler/#1/#2_gremove: }
                  { \use:c {__pdfmanagement_handler/#1/#2_gremove:} } %special handler
                  {
                    \exp_args:Nne
                    \prop_gremove:cn
                      { \__kernel_pdfdict_name:n { g__pdf_Core/#1 } }
                      { \str_convert_pdfname:n {#2} }
                  }
              }
          }
          {
            \msg_error:nnn { pdfmanagement } { unknown-dict } { #1 }
          }
  }

\cs_new_protected:Npn \__pdfmanagement_gremove:nn #1 #2 %path,key
  {
    \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
          {
            \exp_args:Nne
            \prop_gremove:cn
              { \__kernel_pdfdict_name:n { g__pdf_Core/#1 } }
              { \str_convert_pdfname:n{#2} }
          }
          {
            \msg_error:nnn { pdfmanagement } { unknown-dict } { #1 }
          }
  }


\cs_new_protected:Npn \__pdfmanagement_show:Nn #1#2
  {
    \cs_if_exist:cTF
      { __pdfmanagement_handler/#2/?_show: } %general, name independent handler
      { \use:c {__pdfmanagement_handler/#2/?_show:} }
      {
        \prop_if_exist:cTF { \__kernel_pdfdict_name:n { g__pdf_Core/#2 } }
          {
             #1
              { pdfmanagement } { show-dict }
              { \tl_to_str:n {#2} }
              {
                \prop_map_function:cN
                 {\__kernel_pdfdict_name:n { g__pdf_Core/#2 }}
                 \msg_show_item:nn
              }
              { } { }
          }
          {
            #1 { pdfmanagement } { unknown-dict } {#2}{}{}{}
          }
```

```
      }
  }

\cs_new_protected:Npn \__pdfmanagement_show:n #1  %path
  {
    \prop_show:c { \__kernel_pdfdict_name:n { g__pdf_Core/#1 } }
  }
```

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

```
\cs_new_protected:Npn \pdfmanagement_show:n #1
  {
    \__pdfmanagement_show:Nn \msg_show:nneeee {#1}
  }

\cs_new_protected:Npn \pdfmanagement_remove:nn #1 #2
  {
    \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
      {
        \__pdfmanagement_handler_gremove:nn { #1 }{ #2 }
      }
      {
        \msg_error:nnn{pdfmanagement}{unknown-dict}{#1}
      }
  }

\cs_new_protected:Npn \pdfmanagement_get:nnN #1 #2 #3
  {
    \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
      {
        \__pdfmanagement_get:nnN { #1 }{ #2 } #3
      }
      {
        \msg_error:nnn{pdfmanagement}{unknown-dict}{#1}
      }
  }
```

2.4 The Info dictionary

Initialization of the dictionary:

```
\pdfdict_new:n { g__pdf_Core/Info}
```

`\__pdfmanagement_Info_gpush:` `\__pdfmanagement_Info_gpush:` is the command that outputs the info dictionary (currently in the end-of-run hooks).

```
% push to the register command / issue the special
\cs_new_protected:Npn \__pdfmanagement_Info_gpush:
  {
    \prop_map_function:cN
      { \__kernel_pdfdict_name:n { g__pdf_Core/Info} }
      \__pdf_backend_info_gput:nn
    \prop_gclear:c { \__kernel_pdfdict_name:n { g__pdf_Core/Info} }
  }
```

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

2.5 The Pages dictionary code

At first the initialisation

```
\pdfdict_new:n { g__pdf_Core/Pages}
```

__pdfmanagement_Pages_gpush: This is the command that outputs the Pages dictionary. It is used at the end of the document in \g__pdf_backend_end_run_tl

```
% push to the register command / issue the special
\cs_new_protected:Npn \__pdfmanagement_Pages_gpush:
  {
    \pdfdict_if_empty:nF { g__pdf_Core/Pages}
      {
        \exp_args:Ne \__pdf_backend_Pages_primitive:n
          {
            \pdfdict_use:n { g__pdf_Core/Pages}
          }
      }
  }

```

(*End of definition for* __pdfmanagement_Pages_gpush:.)

2.6 The Page and ThisPage dictionary

At first the initialisation.

```
\pdfdict_new:n { g__pdf_Core/Page }
\pdfdict_new:n { g__pdf_Core/ThisPage }

%handler for pdfmanagement
\cs_new_protected:cpn { __pdfmanagement_handler/Page/?_gput:nn } #1 #2
  {
    \__pdf_backend_Page_gput:nn { #1 }{ #2 }
  }
% remove:
\cs_new_protected:cpn { __pdfmanagement_handler/Page/?_gremove:n } #1
  {
    \__pdf_backend_Page_gremove:n { #1 }
  }

% handler for pdfmanagement
\cs_new_protected:cpn { __pdfmanagement_handler/ThisPage/?_gput:nn } #1 #2
  {
    \prop_gput:cnn  { \__kernel_pdfdict_name:n { g__pdf_Core/ThisPage } }{ #1 } { #2 }
    \__pdf_backend_ThisPage_gput:nn { #1 }{ #2 }
  }

\cs_new_protected:cpn { __pdfmanagement_handler/ThisPage/?_gremove:n } #1
  {
    \msg_warning:nnn { pdfmanagement } { no-removal }{ThisPage}
  }

\cs_new_protected:cpn { __pdfmanagement_handler/ThisPage/?_show: }
```

```
  {
    \msg_warning:nnn { pdfmanagement } { no-show }{ThisPage}
  }

```

2.6.1 "Page/Resources": ExtGState, ColorSpace, Shading, Pattern

```
\clist_const:Nn \c__pdfmanagement_PageResources_clist
  {
    ExtGState,
    ColorSpace,
    Pattern,
    Shading,
  }

\clist_map_inline:Nn \c__pdfmanagement_PageResources_clist
  {
    \pdfdict_new:n { g__pdf_Core/Page/Resources/#1}
  }
%
% setter:  #1 is the name of the resource
\cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/ExtGState/?_gput:nn } #1 #2
  {
    \__pdf_backend_PageResources_gput:nnn {ExtGState} { #1 }{ #2 }
  }

\cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/ColorSpace/?_gput:nn } #1 #2
  {
    \__pdf_backend_PageResources_gput:nnn {ColorSpace} { #1 }{ #2 }
  }

\cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/Shading/?_gput:nn } #1 #2
  {
    \__pdf_backend_PageResources_gput:nnn {Shading} { #1 }{ #2 }
  }

\cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/Pattern/?_gput:nn } #1 #2
  {
    \__pdf_backend_PageResources_gput:nnn {Pattern} { #1 }{ #2 }
  }
```

2.6.2 "Catalog"

The catalog has mixed entries: toplevel, subdictionaries, and entries which must build arrays.

\c__pdfmanagement_Catalog_toplevel_clist
\c__pdfmanagement_Catalog_sub_clist
\c__pdfmanagement_Catalog_seq_clist

This variables hold the list of the various types of entries. With it the various `_gput` commands are generated.

(*End of definition for* `\c__pdfmanagement_Catalog_toplevel_clist`, `\c__pdfmanagement_Catalog_sub_clist`, *and* `\c__pdfmanagement_Catalog_seq_clist`.)

__pdfmanagement_catalog_XX_gput:n

Various commands to handle subentries and special cases. At first we set up a few lists of the various types.

```
\pdfdict_new:n { g__pdf_Core/Catalog}

\clist_const:Nn \c__pdfmanagement_Catalog_toplevel_clist
  {
    Collection,
    DPartRoot,
    Lang,
    Legal,
    Metadata,
    NeedsRendering,
    OCProperties/D,
    OpenAction,
    PageLabels,
    PageLayout,
    PageMode,
    Perms,
    PieceInfo,
    SpiderInfo,
    StructTreeRoot,
    Threads,
    URI,
    Version
  }

\clist_const:Nn \c__pdfmanagement_Catalog_sub_clist
  {
    AA,
    AcroForm,
    AcroForm/DR,
    AcroForm/DR/Font,
    MarkInfo,
    ViewerPreferences,
    OCProperties
  }

\clist_map_inline:Nn \c__pdfmanagement_Catalog_sub_clist
  {
    \pdfdict_new:n { g__pdf_Core/Catalog/#1}
  }


\clist_const:Nn \c__pdfmanagement_Catalog_seq_clist
  {
    AF,
    OCProperties/OCGs,
    OCProperties/Configs,
    OutputIntents,
    Requirements,
    AcroForm/Fields,
    AcroForm/CO
  }

```

Names trees in Catalog/Names. We prepare the full list but activate only AP and

JavaScript for now. /EmbeddedFiles has special code and so is not in the name list.

```
\clist_const:Nn \c__pdfmanagement_Catalog_nametree_clist
  {
    AP,
    JavaScript,
%   Pages,
%   Templates,
%   IDS,
%   URLS,
%   Renditions
  }
```

now we create the handler. The entries in the seq-list store in a seq

```
\clist_map_inline:Nn \c__pdfmanagement_Catalog_seq_clist
 {
   \seq_new:c { g__pdfmanagement_/Catalog/#1_seq } % new name later
   \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/#1_gput:n } ##1
     {
       \seq_gput_right:cn { g__pdfmanagement_/Catalog/#1_seq } {  ##1  }
     }
 }

```

OCProperties/D is special: it handles a default. This is done by adding to the left of the seq

```
\cs_new_protected:cpn { __pdfmanagement_handler/Catalog/OCProperties/D_gput:n } #1
  {
    \seq_gput_left:cn
      { g__pdfmanagement_/Catalog/OCProperties/Configs_seq }
      {  #1  }
  }
```

The name tree keys store in a property and check for duplicates. This is done with an auxiliary.

```
\cs_new_protected:Npn \__pdfmanagement_nametree_add_aux:nnn #1 #2 #3
  %#1 name tree, #2 sanitized name #3 value
  {
     \prop_get:coNTF
       { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/Names/#1 }}
       { #2 }
       \l__pdfmanagement_tmpb_tl
       {
         \msg_error:nnnn{pdfmanagement}{name-exist}{#2}{#1}
       }
       {
         \prop_gput:con
           { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/Names/#1 }}
           { #2 }
           { #3 }
       }
  }
```

This is the standard handler for most names trees:

```
\clist_map_inline:Nn \c__pdfmanagement_Catalog_nametree_clist
 {
   \pdfdict_new:n { g__pdf_Core/Catalog/Names/#1}
   \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/Names/#1/?_gput:nn } ##1 ##2
     {
       \pdf_string_from_unicode:nnN {utf8/string}{##1}\l__pdfmanagement_tmpa_tl
       \exp_args:Nno
         \__pdfmanagement_nametree_add_aux:nnn {#1}{\l__pdfmanagement_tmpa_tl}{##2}
     }
 }
```

EmbeddedFiles is a bit special. For once there is special backend code needed by dvips. Beside this we also want the option to create the file name on the fly, so they are actually two access methods: `\pdfmanagement_add:nnn{Catalog/Names/EmbeddedFiles}{name}{reference}` and `\pdfmanagement_add:nnn{Catalog/Names}{EmbeddedFiles}{reference}`

```
\pdfdict_new:n { g__pdf_Core/Catalog/Names/EmbeddedFiles}
\cs_new_protected:cpn { __pdfmanagement_handler/Catalog/Names/EmbeddedFiles/?_gput:nn } #1 #2
  {
    \pdf_string_from_unicode:nnN {utf8/string}{#1}\l__pdfmanagement_tmpa_tl
    \exp_args:Nno
    \__pdfmanagement_nametree_add_aux:nnn
      {EmbeddedFiles}{\l__pdfmanagement_tmpa_tl}{#2}
    \exp_args:No
    \__pdf_backend_NamesEmbeddedFiles_add:nn {\l__pdfmanagement_tmpa_tl}{#2}
  }
```

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

Building the catalog: Push order

`\__pdfmanagement_Catalog_gpush:`

```
\cs_new_protected:Npn \__pdfmanagement_Catalog_gpush:
  {
    \use:c { __pdfmanagement_/Catalog/AA_gpush: }
    \use:c { __pdfmanagement_/Catalog/AcroForm_gpush: }
    \use:c { __pdfmanagement_/Catalog/AF_gpush: }
    \use:c { __pdfmanagement_/Catalog/MarkInfo_gpush: }
    \pdfmeta_standard_verify:nT {Catalog_no_OCProperties}
      {
        \use:c { __pdfmanagement_/Catalog/OCProperties_gpush: }
      }
    \use:c { __pdfmanagement_/Catalog/OutputIntents_gpush: }
    \use:c { __pdfmanagement_/Catalog/Requirements_gpush: }
    \use:c { __pdfmanagement_/Catalog/ViewerPreferences_gpush: }
    % output the single values:
    \prop_map_function:cN
      { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog} }
      \__pdf_backend_catalog_gput:nn
    % output names tree:
    \use:c{ __pdfmanagement_/Catalog/Names_gpush:n } {EmbeddedFiles}
```

```
    \clist_map_inline:Nn \c__pdfmanagement_Catalog_nametree_clist
     {
      \use:c{ __pdfmanagement_/Catalog/Names_gpush:n } {##1}
     }
  }
```

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

Building catalog entries: AA

`\__pdfmanagement_/Catalog/AA_gpush:`

```
\cs_new_protected:cpn { __pdfmanagement_/Catalog/AA_gpush: }
  {
    \prop_if_empty:cF
     { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AA } }
     {
       \pdf_object_new:n  { __pdfmanagement/Catalog/AA }
       \pdf_object_write:nne
            { __pdfmanagement/Catalog/AA }{ dict }
            { \pdfdict_use:n { g__pdf_Core/Catalog/AA } }
       \exp_args:Nne
         \__pdf_backend_catalog_gput:nn
           {AA}
           {
             \pdf_object_ref:n { __pdfmanagement/Catalog/AA }
           }
     }
  }
```

(*End of definition for* `\__pdfmanagement_/Catalog/AA_gpush:`.)

Building catalog entries: AcroForm This is the most complicated case. The entries is build from /Catalog/AcroForm/Fields (array), /Catalog/AcroForm/CO (array), /Catalog/AcroForm/DR/Font (dict), /Catalog/AcroForm/DR (dict), /Catalog/AcroForm

`\__pdfmanagement_/Catalog/AcroForm_gpush:`

```
\cs_new_protected:cpn { __pdfmanagement_/Catalog/AcroForm_gpush: }
  {
    \seq_if_empty:cF { g__pdfmanagement_/Catalog/AcroForm/Fields_seq }
      {
        \pdf_object_new:n  { __pdfmanagement/Catalog/AcroForm/Fields }
        \pdf_object_write:nne
            { __pdfmanagement/Catalog/AcroForm/Fields } { array }
            { \seq_use:cn { g__pdfmanagement_/Catalog/AcroForm/Fields_seq } {~} }
        \exp_args:Nnne
          \prop_gput:cnn %we have to use \prop here to avoid the handler ...
            { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AcroForm } }
            { Fields }
            { \pdf_object_ref:n { __pdfmanagement/Catalog/AcroForm/Fields } }
      }
    \seq_if_empty:cF { g__pdfmanagement_/Catalog/AcroForm/CO_seq }
      {
```

```
      \pdf_object_new:n  { __pdfmanagement/Catalog/AcroForm/CO }
      \pdf_object_write:nne
          { __pdfmanagement/Catalog/AcroForm/CO } { array }
          { \seq_use:cn { g__pdfmanagement_/Catalog/AcroForm/CO_seq } {~} }
      \exp_args:Nnne
        \prop_gput:cnn %we have to use \prop here to avoid the handler ...
          { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AcroForm } }
          { CO }
          { \pdf_object_ref:n { __pdfmanagement/Catalog/AcroForm/CO } }
   }
  \prop_if_empty:cF { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AcroForm/DR/Font}}
    {
      \pdf_object_new:n { __pdfmanagement/Catalog/AcroForm/DR/Font }
      \pdf_object_write:nne
          { __pdfmanagement/Catalog/AcroForm/DR/Font } { dict }
          { \pdfdict_use:n { g__pdf_Core/Catalog/AcroForm/DR/Font } }
      \exp_args:Nnne
        \prop_gput:cnn %we have to use \prop here to avoid the handler ...
          { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AcroForm/DR } }
          { Font }
          { \pdf_object_ref:n { __pdfmanagement/Catalog/AcroForm/DR/Font } }
    }
  \prop_if_empty:cF { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AcroForm/DR}}
    {
      \pdf_object_new:n { __pdfmanagement/Catalog/AcroForm/DR }
      \pdf_object_write:nne
          { __pdfmanagement/Catalog/AcroForm/DR } { dict }
          { \pdfdict_use:n { g__pdf_Core/Catalog/AcroForm/DR } }
      \exp_args:Nnne
        \prop_gput:cnn %we have to use \prop here to avoid the handler ...
          { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AcroForm } }
          { DR }
          { \pdf_object_ref:n { __pdfmanagement/Catalog/AcroForm/DR } }
    }
  \prop_if_empty:cF { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/AcroForm} }
    {
      \pdf_object_new:n { __pdfmanagement/Catalog/AcroForm }
      \pdf_object_write:nne
          { __pdfmanagement/Catalog/AcroForm } { dict }
          { \pdfdict_use:n { g__pdf_Core/Catalog/AcroForm } }
      \exp_args:Nnne
        \__pdfmanagement_handler_gput:nnn
          { Catalog }
          { AcroForm }
          { \pdf_object_ref:n { __pdfmanagement/Catalog/AcroForm } }
    }
}

```

(*End of definition for* `\__pdfmanagement_/Catalog/AcroForm_gpush:`.)

Building catalog entries: AF AF is an array.

`\__pdfmanagement_/Catalog/AF_gpush:`

```
\cs_new_protected:cpn { __pdfmanagement_/Catalog/AF_gpush: }
  {
    \seq_if_empty:cF
     { g__pdfmanagement_/Catalog/AF_seq }
     {
       \pdf_object_new:n  { __pdfmanagement/Catalog/AF }
       \pdf_object_write:nne
            { __pdfmanagement/Catalog/AF } { array }
            { \seq_use:cn { g__pdfmanagement_/Catalog/AF_seq } {~} }
       \exp_args:Nne
         \__pdf_backend_catalog_gput:nn
           {AF}
           {
             \pdf_object_ref:n {__pdfmanagement/Catalog/AF}
           }
     }
  }
```

(*End of definition for* `\__pdfmanagement_/Catalog/AF_gpush:`.)

Building catalog entries: MarkInfo

`\__pdfmanagement_/Catalog/MarkInfo_gpush:`

```
\cs_new_protected:cpn { __pdfmanagement_/Catalog/MarkInfo_gpush: }
  {
    \prop_if_empty:cF
     { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/MarkInfo } }
     {
       \pdf_object_new:n  { __pdfmanagement/Catalog/MarkInfo }
       \pdf_object_write:nne
          { __pdfmanagement/Catalog/MarkInfo } { dict }
          { \pdfdict_use:n { g__pdf_Core/Catalog/MarkInfo } }
       \exp_args:Nne
         \__pdf_backend_catalog_gput:nn
           {MarkInfo}
           {
             \pdf_object_ref:n {__pdfmanagement/Catalog/MarkInfo}
           }
     }
  }
```

(*End of definition for* `\__pdfmanagement_/Catalog/MarkInfo_gpush:`.)

Building catalog entries: OCProperties This is a dictionary with three entries:

/OCGs (required) An array of indirect references, access needed for more than one package.

/D (required) a dict (given as an object name) to the default configuration

/Configs (optional) an array of indirect references to more configurations.

The /D entry is also a config, it is the first of the seq. The overall structure is nested: a dict with arrays.

pdfmanagement/Catalog/OCProperties_gpush:

```
% Catalog/OCProperties: OCGs + D is required
\cs_new_protected:cpn { __pdfmanagement_/Catalog/OCProperties_gpush: }
  {
   \int_compare:nNnT
      {
        ( \seq_count:c { g__pdfmanagement_/Catalog/OCProperties/OCGs_seq }  )*
        ( \seq_count:c { g__pdfmanagement_/Catalog/OCProperties/Configs_seq } )
      }
      >
      { 0 }
      {
        \pdf_object_new:n  { __pdfmanagement/Catalog/OCProperties }
        \seq_gpop_left:cN { g__pdfmanagement_/Catalog/OCProperties/Configs_seq} \l__pdfmanagemen
        \pdf_object_write:nne {__pdfmanagement/Catalog/OCProperties} {dict}
           {
              /OCGs~[ \seq_use:cn { g__pdfmanagement_/Catalog/OCProperties/OCGs_seq } {~} ]
              /D~\l__pdfmanagement_tmpa_tl~
              \seq_if_empty:cF { g__pdfmanagement_/Catalog/OCProperties/Configs_seq }
                {
                  /Configs~
                  [ \seq_use:cn { g__pdfmanagement_/Catalog/OCProperties/Configs_seq} {~} ]
                }
           }
        \exp_args:Nne
          \__pdf_backend_catalog_gput:nn
            { OCProperties }
            { \pdf_object_ref:n {__pdfmanagement/Catalog/OCProperties} }
      }
  }
```

(*End of definition for* `\__pdfmanagement_/Catalog/OCProperties_gpush:`.)

Building catalog entries: OutputIntents OutputIntents is an array.

pdfmanagement_/Catalog/OutputIntents_gpush:

```
\cs_new_protected:cpn { __pdfmanagement_/Catalog/OutputIntents_gpush: }
  {
    \seq_if_empty:cF
     { g__pdfmanagement_/Catalog/OutputIntents_seq }
     {
       \pdf_object_new:n  { __pdfmanagement/Catalog/OutputIntents }
       \pdf_object_write:nne
            { __pdfmanagement/Catalog/OutputIntents } { array }
            { \seq_use:cn { g__pdfmanagement_/Catalog/OutputIntents_seq } {~} }
       \exp_args:Nne
         \__pdf_backend_catalog_gput:nn
           {OutputIntents}
           {
             \pdf_object_ref:n {__pdfmanagement/Catalog/OutputIntents}
           }
     }
  }
```

(*End of definition for* `\__pdfmanagement_/Catalog/OutputIntents_gpush:`.)

Building catalog entries: Requirements

Requirements is an array.

pdfmanagement/Catalog/Requirements_gpush:

```
\cs_new_protected:cpn { __pdfmanagement_/Catalog/Requirements_gpush: }
  {
    \seq_if_empty:cF
     { g__pdfmanagement_/Catalog/Requirements_seq }
     {
       \pdf_object_new:n  { __pdfmanagement/Catalog/Requirements }
       \pdf_object_write:nne
            { __pdfmanagement/Catalog/Requirements } { array }
            { \seq_use:cn { g__pdfmanagement_/Catalog/Requirements_seq } {~} }
       \exp_args:Nne
         \__pdf_backend_catalog_gput:nn
           {Requirements}
           {
             \pdf_object_ref:n { __pdfmanagement/Catalog/Requirements }
           }
     }
  }
```

(*End of definition for* `\__pdfmanagement_/Catalog/Requirements_gpush:`.)

Building catalog entries: ViewerPreferences

anagement_/Catalog/ViewerPreferences_gpush:

```
\cs_new_protected:cpn { __pdfmanagement_/Catalog/ViewerPreferences_gpush: }
  {
    \prop_if_empty:cF
     { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/ViewerPreferences } }
     {
       \pdf_object_new:n  { __pdfmanagement/Catalog/ViewerPreferences }
       \pdf_object_write:nne
            { __pdfmanagement/Catalog/ViewerPreferences } { dict }
            { \pdfdict_use:n { g__pdf_Core/Catalog/ViewerPreferences } }
       \exp_args:Nne
         \__pdf_backend_catalog_gput:nn
           {ViewerPreferences}
           {
             \pdf_object_ref:n {__pdfmanagement/Catalog/ViewerPreferences}
           }
     }
  }
```

(*End of definition for* `\__pdfmanagement_/Catalog/ViewerPreferences_gpush:`.)

Building catalog entries: Names/EmbeddedFiles

\g__pdfmanagement_EmbeddedFiles_int

We want to create names for files on the fly. For this we use an int.

```
\int_new:N \g__pdfmanagement_EmbeddedFiles_int
```

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

__pdfmanagement_EmbeddedFiles_name:

We use the prefix l3ef, and pad numbers below 9999.

```
\cs_new:Npn \__pdfmanagement_EmbeddedFiles_name:
 {
   (
    l3ef
    \int_compare:nNnT {\g__pdfmanagement_EmbeddedFiles_int} < {10}
     {0}
    \int_compare:nNnT {\g__pdfmanagement_EmbeddedFiles_int} < {100}
     {0}
    \int_compare:nNnT {\g__pdfmanagement_EmbeddedFiles_int} < {1000}
     {0}
    \int_use:N \g__pdfmanagement_EmbeddedFiles_int
   )
 }
```

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

_handler/Catalog/Names/EmbeddedFiles_gput:n

EmbeddedFiles is an array and needs a special handler to add values.

```
\pdfdict_new:n { g__pdf_Core/Catalog/Names }

\cs_new_protected:cpn { __pdfmanagement_handler/Catalog/Names/EmbeddedFiles_gput:n } #1
  {
    \int_gincr:N \g__pdfmanagement_EmbeddedFiles_int
    \exp_args:Nne
    \prop_gput:cnn
      { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/Names/EmbeddedFiles }}
      { \__pdfmanagement_EmbeddedFiles_name: }
      { #1 }
     \exp_args:Ne
     \__pdf_backend_NamesEmbeddedFiles_add:nn {\__pdfmanagement_EmbeddedFiles_name:} { #1 }
  }
```

(*End of definition for* `\__pdfmanagement_handler/Catalog/Names/EmbeddedFiles_gput:n`.)

This pushes out the other names trees (but not with dvips). TODO: currently it simply write in the root of the name tree. That is the fastest. If they get longer we perhaps need to build something with Kids and Limits.

__pdfmanagement_/Catalog/Names/?_gpush:

```
\cs_new_protected:cpn  { __pdfmanagement_/Catalog/Names_gpush:n } #1 %#1 name of name tree
 {
   \pdfdict_if_empty:nF { g__pdf_Core/Catalog/Names/#1 }
     {
       \seq_clear:N \l__pdfmanagement_tmpa_seq
```

```
      \prop_map_inline:cn
        {\__kernel_pdfdict_name:n { g__pdf_Core/Catalog/Names/#1 }}
        { \seq_put_right:Nn \l__pdfmanagement_tmpa_seq {##1~##2}}
      \seq_sort:Nn  \l__pdfmanagement_tmpa_seq
        {
          \str_compare:nNnTF {##1} > {##2}
            { \sort_return_swapped: }
            { \sort_return_same: }
        }
      \exp_args:Nne \__pdf_backend_Names_gpush:nn
        {#1}
        {
          \seq_use:Nn \l__pdfmanagement_tmpa_seq {~}
        }
    }
}
```

(*End of definition for* `\__pdfmanagement_/Catalog/Names/?_gpush:n`.)

`__pdfmanagement_handler/Catalog/?_show:` A handler to show the catalog.

```
\cs_new_protected:cpn {__pdfmanagement_handler/Catalog/?_show:}
  {
    \iow_term:e
      {
        \iow_newline:
        The~Catalog~contains~in~the~top~level~the~single~value~entries
        \prop_map_function:cN
          {\__kernel_pdfdict_name:n { g__pdf_Core/Catalog }}
          \msg_show_item:nn
      }
    \clist_map_inline:Nn \c__pdfmanagement_Catalog_seq_clist
     {
       \seq_if_empty:cF {  g__pdfmanagement_/Catalog/##1_seq }
         {
           \iow_term:e
             {
               The~'##1'~array~contains~the~entries
               \seq_map_function:cN { g__pdfmanagement_/Catalog/##1_seq } \msg_show_item:n
             }
         }
      }
    \clist_map_inline:Nn \c__pdfmanagement_Catalog_sub_clist
      {
        \prop_if_empty:cF { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/##1 } }
          {
            \iow_term:e
              {
                The~Catalog~subdirectory~'##1'~contains~the~single~value~entries
                \prop_map_function:cN
                  {\__kernel_pdfdict_name:n { g__pdf_Core/Catalog/##1 }}
                  \msg_show_item:nn
              }
          }
```

```
722       }
723     \tl_show:e {\tl_to_str:n{\pdfmanagement_show:n{Catalog}}}
724   }
```

(*End of definition for* `__pdfmanagement_handler/Catalog/?_show:`.)

2.7 xform / Properties

```
725 \pdfdict_new:n { g__pdf_Core/Xform/Resources/Properties}
```

726 ⟨/package⟩

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.

S

T

U