

The **l3pdfmanagement** module

Managing central PDF resources

L^AT_EX PDF management testphase bundle

The L^AT_EX Project*

Version 0.95n, released 2022-04-14

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

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

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

| | |
|-----------------------------------|--------------------------------|
| Info | <code>\pdfinfo</code> |
| Catalog & various subdictionaries | <code>\pdfcatalog</code> |
| Pages | <code>\pdfpagesattr</code> |
| Page, ThisPage | <code>\pdfpageattr</code> |
| Page/Resources/ExtGState | <code>\pdfpageresources</code> |
| Page/Resources/Shading | <code>\pdfpageresources</code> |
| Page/Resources/Pattern | <code>\pdfpageresources</code> |
| Page/Resources/ColorSpace | <code>\pdfpageresources</code> |

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

| | |
|---|--|
| <code>\pdfmanagement_add:nnn</code> | <code>\pdfmanagement_add:nnn {<resource path>} {<name>} {<value>}</code> |
| <code>\pdfmanagement_add:(nnx nxx xxx)</code> | |
| <code>\PDFManagementAdd</code> | |

New: 2020-04-06

Updated: 2021-07-23

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 dictionary¹. 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:xxx` 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.

| | |
|------------------------------------|--|
| <code>\pdfmanagement_show:n</code> | <code>\pdfmanagement_show:n {<resource path>}</code> |
|------------------------------------|--|

New: 2020-04-08

This shows the content of the dictionary targetted 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.

| | |
|---------------------------------------|--|
| <code>\pdfmanagement_remove:nn</code> | <code>\pdfmanagement_remove:nn {<resource path>} {<name>}</code> |
|---------------------------------------|--|

New: 2020-04-07

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).

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

1.2 Description of the resource pathes

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 | <code>\pdfmanagement_add:nnn {Info} {<name>} {<value>}</code> |
|---------------------|---|

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:nnx {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. 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`.

1.2.4 “Page/Resources”: ExtGState, ColorSpace, Shading, Pattern

```
pdfmanagement: Page/Resources/ExtGState \pdfmanagement_add:nnn {Page/Resources/<resource>} {<name>}
pdfmanagement: Page/Resources/ColorSpace {<value>}
pdfmanagement: Page/Resources/Shading
pdfmanagement: Page/Resources/Pattern
```

Updated: 2020-04-10

Adds `/<name> <value>` to the page resource `<resource>`. `<resource>` can be **ExtGState**, **ColorSpace**, **Pattern** oder **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 pathes 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 `l3pdf file` module. This clashes with e.g. the `embedfile` package which should not be used!

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.

²This is similar to how pgf handles this resources

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 pathes 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 | |
| Catalog/AcroForm/DR | XFA <i><name></i> | stream or array | pdf 1.5 probably unneeded |
| Catalog/AcroForm/DR/Font | <i><name></i> | 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:nn {module/intent}{dict}
\pdf_object_write:nn {module/intent}{...}
\pdfmanagement_add:nxx {Catalog} {OutputIntents}{\pdf_object_ref:n {module/intent}}
```

or

```
\pdf_object_unnamed_write:nn {dict} { ... }
\pdfmanagement_add:nxx {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:nnx { 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 er |
| 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 Unicod |

2 l3pdfmanagement implementation

```

1 <@@=pdfmanagement>
2 <*header>
3 %
4 \ProvidesExplPackage{l3pdfmanagement}{2022-04-14}{0.95n}
5 {Management of core PDF dictionaries (LaTeX PDF management testphase bundle)}
6 </header>

```

2.1 Messages

```

7 <*package>
8 \msg_new:nnn { pdfmanagement } { unknown-dict }
9 { The~PDF~management~resource~'#1'~is~unknown. }
10
11 \msg_new:nnn { pdfmanagement } { empty-value }
12 { The~value~for~'#1'~is~empty~and~will~be~ignored }
13
14 \msg_new:nnn { pdfmanagement } { no-removal }
15 { It~is~not~possible~to~remove~values~from~'#1'~.}
16
17 \msg_new:nnn { pdfmanagement } { no-show }
18 { It~is~not~possible~to~show~the~content~of~'#1'~.}
19
20 \msg_new:nnn { pdfmanagement } { name-exist }
21 { The~name~'#1'~has~already~been~used~for~name~tree~'#2'~.}
22
23 \msg_new:nnn { pdfmanagement } { show-dict }
24 {
25   The~PDF~resource~'#1'~
26   \tl_if_empty:nTF {#2}
27   { is~empty \\>~ . }
28   { contains~the~pairs~(without~outer~braces): #2 . }
29 }
30 \msg_new:nnn { pdfmanagement } { dict-already-defined }
31 {
32   The~path~'#1'~is~already~defined.
33 }
34 \msg_new:nnn { pdfmanagement } { inactive }

```

```

35 {
36   The~PDF~resources~management~is~not~active\\
37   command~'#1'~ignored.
38 }

```

\l__pdfmanagement_tmpa_tl Some temp variables

```

\l__pdfmanagement_tmpb_tl 39 \tl_new:N \l__pdfmanagement_tmpa_tl
\l__pdfmanagement_tmpa_seq 40 \tl_new:N \l__pdfmanagement_tmpb_tl
41 \seq_new:N \l__pdfmanagement_tmpa_seq

```

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

\g__pdfmanagement_active_bool This boolean will control the activation of the management code. It is used in the hooks, and in some backend files. \DocumentMetadata should set it to true

```

42 \bool_new:N \g__pdfmanagement_active_bool

```

(End definition for \g__pdfmanagement_active_bool.)

A user predicate to test if the management code is active

```

43 \prg_new_conditional:Npnn \__pdfmanagement_if_active: { p , T , F , TF }
44 {
45   \bool_if:NTF \g__pdfmanagement_active_bool
46   { \prg_return_true: }
47   { \prg_return_false: }
48 }
49 \prg_set_eq_conditional:NNn
50 \pdfmanagement_if_active: \__pdfmanagement_if_active: { p , T , F , TF }
51
52 \cs_set_eq:NN \IfPDFManagementActiveTF\pdfmanagement_if_active:TF

```

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

```

53 \hook_new:n {pdfmanagement/add}
54 \cs_new_protected:Npn \pdfmanagement_add:nnn #1 #2 #3
55 {
56   \__pdfmanagement_if_active:TF
57   {
58     \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
59     {
60       \hook_gput_code:nnn
61       {pdfmanagement/add}
62       {pdfmanagement}
63       {
64         \__pdfmanagement_handler_gput:nnn { #1 }{ #2 }{ #3 }
65       }
66     }
67     {
68       \msg_error:nnn{pdfmanagement}{unknown-dict}{#1}
69     }
70   }
71   {
72     \msg_warning:nnx {pdfmanagement}{inactive}
73     {\tl_to_str:n {\pdfmanagement_add:nnn}}
74   }
75 }
76

```

```

77 \cs_generate_variant:Nn \pdfmanagement_add:nnn {nnx,nxx,xxx}
78 \cs_set_eq:NN \PDFManagementAdd \pdfmanagement_add:xxx

```

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: pdf_latex/lualatex can execute code after the last `\clearpage` which the dvi-based drivers have to add on a shipout page.

This variables contain the code run in the three places.

```

79 \tl_new:N \g__kernel_pdfmanagement_thispage_shipout_code_tl
80 \tl_new:N \g__kernel_pdfmanagement_lastpage_shipout_code_tl
81 \tl_new:N \g__kernel_pdfmanagement_end_run_code_tl

(End definition for \g__kernel_pdfmanagement_thispage_shipout_code_tl \g__kernel_pdfmanagement_
lastpage_shipout_code_tl \g__kernel_pdfmanagement_end_run_code_tl.)

82 \tl_gset:Nn \g__kernel_pdfmanagement_thispage_shipout_code_tl
83 {
84     \bool_if:NT \g__pdfmanagement_active_bool
85     {
86         \exp_args:NV \__pdf_backend_ThisPage_gpush:n { \g_shipout_readonly_int }
87         \exp_args:NV \__pdf_backend_PageResources_gpush:n { \g_shipout_readonly_int }
88     }
89 }

90
91 \tl_gset:Nn \g__kernel_pdfmanagement_end_run_code_tl
92 {
93     \bool_if:NT \g__pdfmanagement_active_bool
94     {
95         \__pdf_backend_PageResources_obj_gpush: %ExtGState etc
96         \__pdfmanagement_Pages_gpush: %pagesattr
97         \__pdfmanagement_Info_gpush: %pdfinfo
98         \__pdfmanagement_Catalog_gpush:
99     }
100 }

```

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_handler_gput:nnn is the main command to fill the dictionaries. In
__pdfmanagement_get:nnN simple cases it directly fill the property list, but if a handler exists this is called. It is
__pdfmanagement_gremove:nn important to use it only in places where this make sense.
__pdfmanagement_show:n

101
102 %global
103 \cs_new_protected:Npn \__pdfmanagement_handler_gput:nnn #1 #2 #3 % #1 dict, #2 name, #3 value
104 {
105   \tl_if_empty:nTF { #3 }
106   {
107     \msg_none:nnn { pdfmanagement } { empty-value } { /#1/#2 }
108   }
109   {
110     \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
111     {
112       \cs_if_exist:cTF
113       { __pdfmanagement_handler/#1/?_gput:nn } %general, name independant handler
114       { \use:c {__pdfmanagement_handler/#1/?_gput:nn} {#2} {#3} }
115       {
116         \cs_if_exist:cTF
117         { __pdfmanagement_handler/#1/#2_gput:n }
118         { \use:c {__pdfmanagement_handler/#1/#2_gput:n} {#3} } %special handler
119         {
120           \exp_args:Nnx
121           \prop_gput:cnn
122           { \__kernel_pdfdict_name:n { g__pdf_Core/#1 } }
123           { \str_convert_pdfname:n { #2 } }
124           { #3 }
125         }
126       }
127     }
128     {
129       \msg_error:nnn { pdfmanagement } { unknown-dict } { #1 }
130     }
131   }
132 }
133
134
135 \cs_generate_variant:Nn \__pdfmanagement_handler_gput:nnn {nxx}
136
137 \cs_new_protected:Npn \__pdfmanagement_get:nnN #1 #2 #3 %path,key,macro
138 {
139   \exp_args:Nnx

```

```

140     \prop_get:cnN
141     { \__kernel_pdffdict_name:n { g__pdf_Core/#1 } }
142     { \str_convert_pdfname:n {#2} } } #3
143 }
144
145
146 \cs_new_protected:Npn \__pdfmanagement_handler_gremove:nn #1 #2 %path,key
147 {
148     \pdffdict_if_exist:nTF { g__pdf_Core/#1 }
149     {
150         \cs_if_exist:cTF
151         { __pdfmanagement_handler/#1/?_gremove:n } %general, name independant handler
152         { \use:c {__pdfmanagement_handler/#1/?_gremove:n} {#2} }
153         {
154             \cs_if_exist:cTF
155             { __pdfmanagement_handler/#1/#2_gremove: }
156             { \use:c {__pdfmanagement_handler/#1/#2_gremove:} } %special handler
157             {
158                 \exp_args:Nnx
159                 \prop_gremove:cn
160                 { \__kernel_pdffdict_name:n { g__pdf_Core/#1 } }
161                 { \str_convert_pdfname:n {#2} }
162             }
163         }
164     }
165     {
166         \msg_error:nnn { pdfmanagement } { unknown-dict } { #1 }
167     }
168 }
169
170 \cs_new_protected:Npn \__pdfmanagement_gremove:nn #1 #2 %path,key
171 {
172     \pdffdict_if_exist:nTF { g__pdf_Core/#1 }
173     {
174         \exp_args:Nnx
175         \prop_gremove:cn
176         { \__kernel_pdffdict_name:n { g__pdf_Core/#1 } }
177         { \str_convert_pdfname:n{#2} }
178     }
179     {
180         \msg_error:nnn { pdfmanagement } { unknown-dict } { #1 }
181     }
182 }
183
184
185 \cs_new_protected:Npn \__pdfmanagement_show:Nn #1#2
186 {
187     \cs_if_exist:cTF
188     { __pdfmanagement_handler/#2/?_show: } %general, name independant handler
189     { \use:c {__pdfmanagement_handler/#2/?_show:} }
190     {
191         \prop_if_exist:cTF { \__kernel_pdffdict_name:n { g__pdf_Core/#2 } }
192         {
193             #1

```

```

194         { pdfmanagement } { show-dict }
195         { \tl_to_str:n {#2} }
196         {
197             \prop_map_function:cN
198             { \__kernel_pdffdict_name:n { g__pdf_Core/#2 } }
199             \msg_show_item:nn
200         }
201         { } { }
202     }
203     {
204         #1 { pdfmanagement } { unknown-dict } {#2}{-}{-}{-}
205     }
206 }
207 }
208
209 \cs_new_protected:Npn \__pdfmanagement_show:n #1 %path
210 {
211     \prop_show:c { \__kernel_pdffdict_name:n { g__pdf_Core/#1 } }
212 }
213
214 (End definition for \__pdfmanagement_handler_gput:nnn and others.)
215
216 \cs_new_protected:Npn \pdfmanagement_show:n #1
217 {
218     {
219         \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
220         {
221             \__pdfmanagement_handler_gremove:nn { #1 } { #2 }
222         }
223         {
224             \msg_error:nnn{pdfmanagement}{unknown-dict}{#1}
225         }
226     }
227
228 \cs_new_protected:Npn \pdfmanagement_get:nnN #1 #2 #3
229 {
230     {
231         \pdfdict_if_exist:nTF { g__pdf_Core/#1 }
232         {
233             \__pdfmanagement_get:nnN { #1 } { #2 } { #3 }
234         }
235         {
236             \msg_error:nnn{pdfmanagement}{unknown-dict}{#1}
237         }
238     }
239 }

```

2.4 The Info dictionary

Initialization of the dictionary:

```
237 \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).

```

238 % push to the register command / issue the special
239 \cs_new_protected:Npn \__pdfmanagement_Info_gpush:
240 {
241     \prop_map_function:cN
242     { \__kernel_pdffdict_name:n { g__pdf_Core/Info} }
243     \__pdf_backend_info_gput:nn
244     \prop_gclear:c { \__kernel_pdffdict_name:n { g__pdf_Core/Info} }
245 }

```

(End definition for __pdfmanagement_Info_gpush:.)

2.5 The Pages dictionary code

At first the initialisation

```

246 \pdffdict_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

```

247 % push to the register command / issue the special
248 \cs_new_protected:Npn \__pdfmanagement_Pages_gpush:
249 {
250     \pdffdict_if_empty:nF { g__pdf_Core/Pages}
251     {
252         \exp_args:Nx \__pdf_backend_Pages_primitive:n
253         {
254             \pdffdict_use:n { g__pdf_Core/Pages}
255         }
256     }
257 }
258

```

(End definition for __pdfmanagement_Pages_gpush:.)

2.6 The Page and ThisPage dictionary

At first the initialisation.

```

259 \pdffdict_new:n { g__pdf_Core/Page }
260 \pdffdict_new:n { g__pdf_Core/ThisPage }
261
262 %handler for pdfmanagement
263 \cs_new_protected:cpn { __pdfmanagement_handler/Page/?_gput:nn } #1 #2
264 {
265     \__pdf_backend_Page_gput:nn { #1 } { #2 }
266 }
267 % remove:
268 \cs_new_protected:cpn { __pdfmanagement_handler/Page/?_gremove:n } #1
269 {
270     \__pdf_backend_Page_gremove:n { #1 }
271 }
272
273 % handler for pdfmanagement
274 \cs_new_protected:cpn { __pdfmanagement_handler/ThisPage/?_gput:nn } #1 #2
275 {
276     \prop_gput:cnn { \__kernel_pdffdict_name:n { g__pdf_Core/ThisPage } } { #1 } { #2 }

```

```

277     \bool_if:NT \g__pdfmanagement_active_bool
278     {
279         \__pdf_backend_ThisPage_gput:nn { #1 }{ #2 }
280     }
281 }
282
283 \cs_new_protected:cpn { __pdfmanagement_handler/ThisPage/?_gremove:n } #1
284 {
285     \msg_warning:nnn { pdfmanagement } { no-removal }{ThisPage}
286 }
287
288 \cs_new_protected:cpn { __pdfmanagement_handler/ThisPage/?_show: }
289 {
290     \msg_warning:nnn { pdfmanagement } { no-show }{ThisPage}
291 }
292

```

2.6.1 “Page/Resources”: ExtGState, ColorSpace, Shading, Pattern

```

293 \clist_const:Nn \c__pdfmanagement_PageResources_clist
294 {
295     ExtGState,
296     ColorSpace,
297     Pattern,
298     Shading,
299 }
300
301 \clist_map_inline:Nn \c__pdfmanagement_PageResources_clist
302 {
303     \pdfdict_new:n { g__pdf_Core/Page/Resources/#1}
304 }
305 %
306 % setter: #1 is the name of the resource
307 \cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/ExtGState/?_gput:nn } #1 #2
308 {
309     \__pdf_backend_PageResources_gput:nnn {ExtGState} { #1 }{ #2 }
310 }
311
312 \cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/ColorSpace/?_gput:nn } #1 #2
313 {
314     \__pdf_backend_PageResources_gput:nnn {ColorSpace} { #1 }{ #2 }
315 }
316
317 \cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/Shading/?_gput:nn } #1 #2
318 {
319     \__pdf_backend_PageResources_gput:nnn {Shading} { #1 }{ #2 }
320 }
321
322 \cs_new_protected:cpn { __pdfmanagement_handler/Page/Resources/Pattern/?_gput:nn } #1 #2
323 {
324     \__pdf_backend_PageResources_gput:nnn {Pattern} { #1 }{ #2 }
325 }

```

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 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.

```

326 \pdfdict_new:n { g__pdf_Core/Catalog}
327
328 \clist_const:Nn \c_pdfmanagement_Catalog_toplevel_clist
329 {
330     Collection,
331     DPartRoot,
332     Lang,
333     Legal,
334     Metadata,
335     NeedsRendering,
336     OCProperties/D,
337     OpenAction,
338     PageLabels,
339     PageLayout,
340     PageMode,
341     Perms,
342     PieceInfo,
343     SpiderInfo,
344     StructTreeRoot,
345     Threads,
346     URI,
347     Version
348 }
349
350 \clist_const:Nn \c_pdfmanagement_Catalog_sub_clist
351 {
352     AA,
353     AcroForm,
354     AcroForm/DR,
355     AcroForm/DR/Font,
356     MarkInfo,
357     ViewerPreferences,
358     OCProperties
359 }
360
361 \clist_map_inline:Nn \c_pdfmanagement_Catalog_sub_clist
362 {
363     \pdfdict_new:n { g__pdf_Core/Catalog/#1}
364 }
365
366
367 \clist_const:Nn \c_pdfmanagement_Catalog_seq_clist
```

```

368 {
369     AF,
370     OCPProperties/OCGs,
371     OCPProperties/Configs,
372     OutputIntents,
373     Requirements,
374     AcroForm/Fields,
375     AcroForm/CO
376 }
377

```

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.

```

378 \clist_const:Nn \c__pdfmanagement_Catalog_nametree_clist
379 {
380     AP,
381     JavaScript,
382     % Pages,
383     % Templates,
384     % IDS,
385     % URLS,
386     % Renditions
387 }

```

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

```

388 \clist_map_inline:Nn \c__pdfmanagement_Catalog_seq_clist
389 {
390     \seq_new:c { g__pdfmanagement_/Catalog/#1_seq } % new name later
391     \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/#1_gput:n } ##1
392     {
393         \seq_gput_right:cn { g__pdfmanagement_/Catalog/#1_seq } { ##1 }
394     }
395 }
396

```

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

```

397 \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/OCPProperties/D_gput:n } #1
398 {
399     \seq_gput_left:cn
400     { g__pdfmanagement_/Catalog/OCPProperties/Configs_seq }
401     { #1 }
402 }

```

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

```

403 \cs_new_protected:Npn \__pdfmanagement_nametree_add_aux:nnn #1 #2 #3
404     %#1 name tree, #2 sanitized name #3 value
405     {
406         \prop_get:coNTF
407         { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/Names/#1 }}
408         { #2 }
409         \l__pdfmanagement_tmpb_tl
410         {
411             \msg_error:nnnn{pdfmanagement}{name-exist}{#2}{#1}

```

```

412     }
413     {
414         \prop_gput:con
415         { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/Names/#1 }}
416         { #2 }
417         { #3 }
418     }
419 }

```

This is the standard handler for most names trees:

```

420 \clist_map_inline:Nn \c__pdfmanagement_Catalog_nametree_clist
421 {
422     \pdffdict_new:n { g__pdf_Core/Catalog/Names/#1}
423     \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/Names/#1/?_gput:nn } ##1 ##2
424     {
425         \pdf_string_from_unicode:nnN {utf8/string}{##1}\l__pdfmanagement_tmpa_tl
426         \exp_args:Nno
427         \__pdfmanagement_nametree_add_aux:nnn {#1}{\l__pdfmanagement_tmpa_tl}{##2}
428     }
429 }

```

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}`

```

430 \pdffdict_new:n { g__pdf_Core/Catalog/Names/EmbeddedFiles}
431 \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/Names/EmbeddedFiles/?_gput:nn } #1 #2
432 {
433     \pdf_string_from_unicode:nnN {utf8/string}{#1}\l__pdfmanagement_tmpa_tl
434     \exp_args:Nno
435     \__pdfmanagement_nametree_add_aux:nnn
436     {EmbeddedFiles}{\l__pdfmanagement_tmpa_tl}{#2}
437     \exp_args:No
438     \__pdf_backend_NamesEmbeddedFiles_add:nn {\l__pdfmanagement_tmpa_tl}{#2}
439 }

```

(End definition for `__pdfmanagement_catalog_XX_gput:n`.)

Building the catalog: Push order

`__pdfmanagement_Catalog_gpush:`

```

440 \cs_new_protected:Npn \__pdfmanagement_Catalog_gpush:
441 {
442     \use:c { __pdfmanagement_/Catalog/AA_gpush: }
443     \use:c { __pdfmanagement_/Catalog/AcroForm_gpush: }
444     \use:c { __pdfmanagement_/Catalog/AF_gpush: }
445     \use:c { __pdfmanagement_/Catalog/MarkInfo_gpush: }
446     \pdfmeta_standard_verify:nT {Catalog_no_OCProperties}
447     {
448         \use:c { __pdfmanagement_/Catalog/OCProperties_gpush: }
449     }
450     \use:c { __pdfmanagement_/Catalog/OutputIntents_gpush: }
451     \use:c { __pdfmanagement_/Catalog/Requirements_gpush: }
452     \use:c { __pdfmanagement_/Catalog/ViewerPreferences_gpush: }
453     % output the single values:

```

```

454 \prop_map_function:cn
455   { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog} }
456   \__pdf_backend_catalog_gput:nn
457   % output names tree:
458   \use:c{ __pdfmanagement_/Catalog/Names_gpush:n } {EmbeddedFiles}
459   \clist_map_inline:Nn \c__pdfmanagement_Catalog_nametree_clist
460   {
461     \use:c{ __pdfmanagement_/Catalog/Names_gpush:n } {##1}
462   }
463 }

```

(End definition for __pdfmanagement_Catalog_gpush:.)

Building catalog entries: AA

__pdfmanagement_/Catalog/AA_gpush:

```

464 \cs_new_protected:cpn { __pdfmanagement_/Catalog/AA_gpush: }
465 {
466   \prop_if_empty:cF
467   { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AA } }
468   {
469     \__pdf_backend_object_new:nn { __pdfmanagement_/Catalog/AA } { dict }
470     \__pdf_backend_object_write:nx
471       { __pdfmanagement_/Catalog/AA }
472       { \pdffdict_use:n { g__pdf_Core/Catalog/AA } }
473     \exp_args:Nnx
474       \__pdf_backend_catalog_gput:nn
475       {AA}
476       {
477         \__pdf_backend_object_ref:n { __pdfmanagement_/Catalog/AA }
478       }
479   }
480 }

```

(End 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:

```

481 \cs_new_protected:cpn { __pdfmanagement_/Catalog/AcroForm_gpush: }
482 {
483   \seq_if_empty:cF { g__pdfmanagement_/Catalog/AcroForm/Fields_seq }
484   {
485     \__pdf_backend_object_new:nn { __pdfmanagement_/Catalog/AcroForm/Fields } { array }
486     \__pdf_backend_object_write:nx
487       { __pdfmanagement_/Catalog/AcroForm/Fields }
488       { \seq_use:cn { g__pdfmanagement_/Catalog/AcroForm/Fields_seq } {-} }
489     \exp_args:Nnnx
490     \prop_gput:cnn %we have to use \prop here to avoid the handler ...
491     { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AcroForm } }
492     { Fields }
493     { \__pdf_backend_object_ref:n { __pdfmanagement_/Catalog/AcroForm/Fields } }

```

```

494     }
495     \seq_if_empty:cF { g__pdfmanagement_/Catalog/AcroForm/CO_seq }
496     {
497         \__pdf_backend_object_new:nn { __pdfmanagement/Catalog/AcroForm/CO } { array }
498         \exp_args:Nnx
499         \__pdf_backend_object_write:nn
500         { __pdfmanagement/Catalog/AcroForm/CO }
501         { \seq_use:cn { g__pdfmanagement_/Catalog/AcroForm/CO_seq } {-} }
502         \exp_args:Nnnx
503         \prop_gput:cnn %we have to use \prop here to avoid the handler ...
504         { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AcroForm } }
505         { CO }
506         { \__pdf_backend_object_ref:n { __pdfmanagement/Catalog/AcroForm/CO } }
507     }
508     \prop_if_empty:cF { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AcroForm/DR/Font}}
509     {
510         \__pdf_backend_object_new:nn { __pdfmanagement/Catalog/AcroForm/DR/Font } {dict}
511         \exp_args:Nnx
512         \__pdf_backend_object_write:nn
513         { __pdfmanagement/Catalog/AcroForm/DR/Font }
514         { \pdffdict_use:n { g__pdf_Core/Catalog/AcroForm/DR/Font } }
515         \exp_args:Nnnx
516         \prop_gput:cnn %we have to use \prop here to avoid the handler ...
517         { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AcroForm/DR } }
518         { Font }
519         { \__pdf_backend_object_ref:n { __pdfmanagement/Catalog/AcroForm/DR/Font } }
520     }
521     \prop_if_empty:cF { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AcroForm/DR}}
522     {
523         \__pdf_backend_object_new:nn { __pdfmanagement/Catalog/AcroForm/DR } {dict}
524         \exp_args:Nnx
525         \__pdf_backend_object_write:nn
526         { __pdfmanagement/Catalog/AcroForm/DR }
527         { \pdffdict_use:n { g__pdf_Core/Catalog/AcroForm/DR } }
528         \exp_args:Nnnx
529         \prop_gput:cnn %we have to use \prop here to avoid the handler ...
530         { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AcroForm } }
531         { DR }
532         { \__pdf_backend_object_ref:n { __pdfmanagement/Catalog/AcroForm/DR } }
533     }
534     \prop_if_empty:cF { \__kernel_pdffdict_name:n { g__pdf_Core/Catalog/AcroForm} }
535     {
536         \__pdf_backend_object_new:nn { __pdfmanagement/Catalog/AcroForm } {dict}
537         \exp_args:Nnx
538         \__pdf_backend_object_write:nn
539         { __pdfmanagement/Catalog/AcroForm }
540         { \pdffdict_use:n { g__pdf_Core/Catalog/AcroForm } }
541         \exp_args:Nnnx
542         \__pdfmanagement_handler_gput:nnn
543         { Catalog }
544         { AcroForm }
545         { \__pdf_backend_object_ref:n { __pdfmanagement/Catalog/AcroForm } }
546     }
547 }

```

548

(End definition for _pdfmanagement_/Catalog/AcroForm_gpush:.)

Building catalog entries: AF AF is an array.

pdfmanagement/Catalog/AF_gpush:

```
549 \cs_new_protected:cpn { __pdfmanagement_/Catalog/AF_gpush: }
550 {
551   \seq_if_empty:cF
552   { g__pdfmanagement_/Catalog/AF_seq }
553   {
554     \__pdf_backend_object_new:nn { __pdfmanagement_/Catalog/AF } { array }
555     \exp_args:Nnx
556       \__pdf_backend_object_write:nn
557         { __pdfmanagement_/Catalog/AF }
558         { \seq_use:cn { g__pdfmanagement_/Catalog/AF_seq } {~} }
559     \exp_args:Nnx
560     \__pdf_backend_catalog_gput:nn
561       {AF}
562       {
563         \__pdf_backend_object_ref:n {__pdfmanagement_/Catalog/AF}
564       }
565   }
566 }
```

(End definition for _pdfmanagement_/Catalog/AF_gpush:.)

Building catalog entries: MarkInfo

pdfmanagement/Catalog/MarkInfo_gpush:

```
567 \cs_new_protected:cpn { __pdfmanagement_/Catalog/MarkInfo_gpush: }
568 {
569   \prop_if_empty:cF
570   { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/MarkInfo } }
571   {
572     \__pdf_backend_object_new:nn { __pdfmanagement_/Catalog/MarkInfo } { dict }
573     \exp_args:Nnx
574       \__pdf_backend_object_write:nn
575         { __pdfmanagement_/Catalog/MarkInfo }
576         { \pdfdict_use:n { g__pdf_Core/Catalog/MarkInfo } }
577     \exp_args:Nnx
578     \__pdf_backend_catalog_gput:nn
579       {MarkInfo}
580       {
581         \__pdf_backend_object_ref:n {__pdfmanagement_/Catalog/MarkInfo}
582       }
583   }
584 }
```

(End definition for _pdfmanagement_/Catalog/MarkInfo_gpush:.)

Building catalog entries: OCPProperties 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/OCPProperties_gpush:

```

585 % Catalog/OCPProperties: OCGs + D is required
586 \cs_new_protected:cpn { __pdfmanagement_/Catalog/OCPProperties_gpush: }
587 {
588   \int_compare:nNtT
589     {
590       ( \seq_count:c { g__pdfmanagement_/Catalog/OCPProperties/OCGs_seq } ) *
591       ( \seq_count:c { g__pdfmanagement_/Catalog/OCPProperties/Configs_seq } )
592     }
593     >
594     { 0 }
595     {
596       \__pdf_backend_object_new:nn { __pdfmanagement_/Catalog/OCPProperties } { dict }
597       \seq_gpop_left:cN { g__pdfmanagement_/Catalog/OCPProperties/Configs_seq } \l__pdfmanag
598       \exp_args:Nnx
599       \__pdf_backend_object_write:nn { __pdfmanagement_/Catalog/OCPProperties }
600       {
601         /OCGs~ [ \seq_use:cn { g__pdfmanagement_/Catalog/OCPProperties/OCGs_seq } {~} ]
602         /D~ \l__pdfmanagement_tmpa_tl~
603         \seq_if_empty:cF { g__pdfmanagement_/Catalog/OCPProperties/Configs_seq }
604         {
605           /Configs~
606           [ \seq_use:cn { g__pdfmanagement_/Catalog/OCPProperties/Configs_seq } {~} ]
607         }
608       }
609       \exp_args:Nnx
610       \__pdf_backend_catalog_gput:nn
611       { OCPProperties }
612       { \__pdf_backend_object_ref:n { __pdfmanagement_/Catalog/OCPProperties } }
613     }
614 }

```

(End definition for __pdfmanagement_/Catalog/OCPProperties_gpush:.)

Building catalog entries: OutputIntents OutputIntents is an array.

__pdfmanagement_/Catalog/OutputIntents_gpush:

```

615 \cs_new_protected:cpn { __pdfmanagement_/Catalog/OutputIntents_gpush: }
616 {
617   \seq_if_empty:cF
618   { g__pdfmanagement_/Catalog/OutputIntents_seq }
619   {
620     \__pdf_backend_object_new:nn { __pdfmanagement_/Catalog/OutputIntents } { array }

```

```

621     \exp_args:Nnx
622     \__pdf_backend_object_write:nn
623     { __pdfmanagement/Catalog/OutputIntents }
624     { \seq_use:cn { g__pdfmanagement_/Catalog/OutputIntents_seq } {~} }
625   \exp_args:Nnx
626   \__pdf_backend_catalog_gput:nn
627   {OutputIntents}
628   {
629     \__pdf_backend_object_ref:n {__pdfmanagement/Catalog/OutputIntents}
630   }
631 }
632 }

```

(End definition for __pdfmanagement_/Catalog/OutputIntents_gpush:.)

Building catalog entries: Requirements Requirements is an array.

__pdfmanagement_/Catalog/Requirements_gpush:

```

633 \cs_new_protected:cpn { __pdfmanagement_/Catalog/Requirements_gpush: }
634 {
635   \seq_if_empty:cF
636   { g__pdfmanagement_/Catalog/Requirements_seq }
637   {
638     \__pdf_backend_object_new:nn { __pdfmanagement/Catalog/Requirements } { array }
639     \exp_args:Nnx
640     \__pdf_backend_object_write:nn
641     { __pdfmanagement/Catalog/Requirements }
642     { \seq_use:cn { g__pdfmanagement_/Catalog/Requirements_seq } {~} }
643     \exp_args:Nnx
644     \__pdf_backend_catalog_gput:nn
645     {Requirements}
646     {
647       \__pdf_backend_object_ref:n { __pdfmanagement/Catalog/Requirements }
648     }
649   }
650 }

```

(End definition for __pdfmanagement_/Catalog/Requirements_gpush:.)

Building catalog entries: ViewerPreferences

__pdfmanagement_/Catalog/ViewerPreferences_gpush:

```

651 \cs_new_protected:cpn { __pdfmanagement_/Catalog/ViewerPreferences_gpush: }
652 {
653   \prop_if_empty:cF
654   { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/ViewerPreferences } }
655   {
656     \__pdf_backend_object_new:nn { __pdfmanagement/Catalog/ViewerPreferences } { dict }
657     \exp_args:Nnx
658     \__pdf_backend_object_write:nn
659     { __pdfmanagement/Catalog/ViewerPreferences }
660     { \pdfdict_use:n { g__pdf_Core/Catalog/ViewerPreferences } }
661     \exp_args:Nnx
662     \__pdf_backend_catalog_gput:nn

```

```

663         {ViewerPreferences}
664     {
665         \__pdf_backend_object_ref:n {__pdfmanagement/Catalog/ViewerPreferences}
666     }
667 }
668 }

```

(End 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.

```

669 \int_new:N \g__pdfmanagement_EmbeddedFiles_int

```

(End definition for \g__pdfmanagement_EmbeddedFiles_int.)

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

```

670 \cs_new:Npn \__pdfmanagement_EmbeddedFiles_name:
671 {
672     (
673         l3ef
674         \int_compare:nNtT {\g__pdfmanagement_EmbeddedFiles_int} < {10}
675         {0}
676         \int_compare:nNtT {\g__pdfmanagement_EmbeddedFiles_int} < {100}
677         {0}
678         \int_compare:nNtT {\g__pdfmanagement_EmbeddedFiles_int} < {1000}
679         {0}
680         \int_use:N \g__pdfmanagement_EmbeddedFiles_int
681     )
682 }

```

(End definition for __pdfmanagement_EmbeddedFiles_name:.)

_handler/Catalog/Names/EmbeddedFiles_gput:n EmbeddedFiles is an array and needs a special handler to add values.

```

683 \pdfdict_new:n { g__pdf_Core/Catalog/Names }
684
685 \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/Names/EmbeddedFiles_gput:n } #1
686 {
687     \int_gincr:N \g__pdfmanagement_EmbeddedFiles_int
688     \exp_args:Nnx
689     \prop_gput:cnn
690     { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/Names/EmbeddedFiles }}
691     { \__pdfmanagement_EmbeddedFiles_name: }
692     { #1 }
693     \exp_args:Nx
694     \__pdf_backend_NamesEmbeddedFiles_add:nn {\__pdfmanagement_EmbeddedFiles_name:} { #1 }
695 }

```

(End 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:

```
696 \cs_new_protected:cpn { __pdfmanagement_/Catalog/Names_gpush:n } #1 % #1 name of name tree
697 {
698   \pdfdict_if_empty:nF { g__pdf_Core/Catalog/Names/#1 }
699   {
700     \seq_clear:N \l__pdfmanagement_tmpa_seq
701     \prop_map_inline:cn
702       { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/Names/#1 } }
703     { \seq_put_right:Nn \l__pdfmanagement_tmpa_seq { ##1~##2 } }
704     \seq_sort:Nn \l__pdfmanagement_tmpa_seq
705     {
706       \str_compare:nNnTF { ##1 } > { ##2 }
707       { \sort_return_swapped: }
708       { \sort_return_same: }
709     }
710     \exp_args:Nnx \__pdf_backend_Names_gpush:nn
711       { #1 }
712     {
713       \seq_use:Nn \l__pdfmanagement_tmpa_seq { ~ }
714     }
715   }
716 }
```

(End definition for __pdfmanagement_/Catalog/Names/?_gpush:.)

__pdfmanagement_handler/Catalog/?_show:

A handler to show the catalog.

```
717 \cs_new_protected:cpn { __pdfmanagement_handler/Catalog/?_show: }
718 {
719   \iow_term:x
720   {
721     \iow_newline:
722     The~Catalog~contains~in~the~top-level~the~single~value~entries
723     \prop_map_function:cN
724       { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog } }
725     \msg_show_item:nn
726   }
727   \clist_map_inline:Nn \c__pdfmanagement_Catalog_seq_clist
728   {
729     \seq_if_empty:cF { g__pdfmanagement_/Catalog/##1_seq }
730     {
731       \iow_term:x
732       {
733         The~'##1'~array~contains~the~entries
734         \seq_map_function:cN { g__pdfmanagement_/Catalog/##1_seq } \msg_show_item:n
735       }
736     }
737   }
738   \clist_map_inline:Nn \c__pdfmanagement_Catalog_sub_clist
739   {
740     \prop_if_empty:cF { \__kernel_pdfdict_name:n { g__pdf_Core/Catalog/##1 } }
741     {
742       \iow_term:x
743       {
744         The~Catalog~subdirectory~'##1'~contains~the~single~value~entries
```

```

745         \prop_map_function:cN
746         {\__kernel_pdffdict_name:n { g__pdf_Core/Catalog/##1 }}
747         \msg_show_item:nn
748     }
749 }
750 }
751 \tl_show:x {\tl_to_str:n{\pdfmanagement_show:n{Catalog}}}
752 }

```

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

2.7 xform / Properties

```

753 \pdffdict_new:n { g__pdf_Core/Xform/Resources/Properties}
754 \</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.

| Symbols | |
|--|-----------------------------------|
| <code>\\</code> | 27, 36 |
| B | |
| bool commands: | |
| <code>\bool_if:NTF</code> | 45, 84, 93, 277 |
| <code>\bool_new:N</code> | 42 |
| C | |
| <code>\clearpage</code> | 12 |
| clist commands: | |
| <code>\clist_const:Nn</code> | 293, 328, 350, 367, 378 |
| <code>\clist_map_inline:Nn</code> | 301, 361, 388, 420, 459, 727, 738 |
| cs commands: | |
| <code>\cs_generate_variant:Nn</code> | 77, 135 |
| <code>\cs_if_exist:NTF</code> | 112, 116, 150, 154, 187 |
| <code>\cs_new:Npn</code> | 670 |
| <code>\cs_new_protected:Npn</code> .. | 54, 103, |
| 137, 146, 170, 185, 209, 213, 217, | |
| 227, 239, 248, 263, 268, 274, 283, | |
| 288, 307, 312, 317, 322, 391, 397, | |
| 403, 423, 431, 440, 464, 481, 549, | |
| 567, 586, 615, 633, 651, 685, 696, 717 | |
| <code>\cs_set_eq:NN</code> | 52, 78 |
| D | |
| <code>\DocumentMetadata</code> | 11 |
| E | |
| exp commands: | |
| <code>\exp_args:Nnnx</code> | 489, 502, 515, 528, 541 |
| <code>\exp_args:Nno</code> | 426, 434 |
| <code>\exp_args:Nnx</code> | 120, |
| 139, 158, 174, 473, 498, 511, 524, | |
| 537, 555, 559, 573, 577, 598, 609, | |
| 621, 625, 639, 643, 657, 661, 688, 710 | |
| <code>\exp_args:No</code> | 437 |
| <code>\exp_args:NW</code> | 86, 87 |
| <code>\exp_args:Nx</code> | 252, 693 |
| H | |
| hook commands: | |
| <code>\hook_gput_code:nnn</code> | 60 |
| <code>\hook_new:n</code> | 53 |
| I | |
| <code>\IfPDFManagementActiveTF</code> | 2, 52 |
| int commands: | |
| <code>\int_compare:nNnTF</code> .. | 588, 674, 676, 678 |
| <code>\int_gincr:N</code> | 687 |
| <code>\int_new:N</code> | 669 |
| <code>\int_use:N</code> | 680 |
| iow commands: | |
| <code>\iow_newline:</code> | 721 |
| <code>\iow_term:n</code> | 719, 731, 742 |
| K | |
| kernel internal commands: | |
| <code>__kernel_pdffdict_name:n</code> | 122, 141, 160, 176, 191, 198, |
| 211, 242, 244, 276, 407, 415, 455, | |

U