The l3pdffile module
Embedding and referencing files in a PDF
LaTeX PDF management bundle

The LaTeX Project[*]

Version 0.96v, released 2025-08-05

1 l3pdffile documentation

1.1 Introduction

1.1.1 Background

External files can be referenced from a PDF in three ways:

1. through an annotation of type Link,

2. by referencing a local file in the file system,

3. by embedding the file directly into the PDF

Case 1 (Links) are created with the `\pdfannot` commands. This module handles the two other cases. Actually from the view of the PDF format they are quite similar: Case 2 is case 3 without the stream object and without the /EF entry in the /Filespec dictionary (this points to the stream object of the file). Not embedding the file makes the PDF smaller. But it is also less portable: the files can only be found if they are in the right location relative to the PDF. The normal case is to embed the file.

The tasks to embed and reference such a file are

1. Embed the file in a stream.

2. Create a Filespec dictionary which references the stream object in the /EF dictionary:

```
<<
 /Type /Filespec
 /F  (l3pdffile.dtx)
 /UF (l3pdffile.dtx)
 /AFRelationship /Source
 /EF <</F 21 0 R /UF 21 0 R>>   %case 3, embedded file
>>
```

---

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

The file names in the /UF and /F value don't need to be identical to the name of file on the disc. It is quite possible to embed a `zzz.tex` and name it `blub.tex`. The second name is then what the user will see in the attachment list or in the properties of an annotation.

3. Reference the Filespec dictionary so that the user can access the file. This can be done in various way:

   (a) With an annotation (/Subtype/FileAttachment). This is done by `attachfile`, `attachfile2` and `intopdf`. Typical entries of such an annotation are:

key	value type	notes
/FS	object reference	(Filespec dictionary)
/Name	name	/Graph, /PushPin, /Paperclip, /Tag
/Contents	text string	optional but recommended
/F	integer	Flags
/AP	dictionary	Appearance (required if rectangle >0)
/AS	name	

   The `/AP` takes precedence over Border and similar keys.

   (b) Through an entry in the `/EmbeddedFiles` name tree. This is what `embedfiles` does.

   ```
   20 0 obj %Document Name tree
    <</EmbeddedFiles 21 0 R>>
        endobj
     21 0 obj %Embedded Files Name dictionary
    <</Names [(AcmeCustomCrypto Protected PDF.pdf) 17 0 R]>>
    endobj
   ```

   The strings (keys) in the `/Names` dictionary must be sorted lexically. But they don't have to be the file name or anything related to the file name. The resource management code uses l3ef0001, l3ef0002 …, which allows up to 9999 files. The key can be needed to identify the start file in a collection, so their relation to the files are stored in a property list.

   (c) Through the `/AF` key in various objects (pdf 2.0). The value is normally an array of object references, but it can also be a name which is mapped to an array in /Properties:

   ```
   /AF /NamedAF BDC
   /Properties <</NamedAF [12 0 R]
   ```

   The related `/Filespec` dictionary should contain an `/AFRelationship` key in this case (but it doesn't harm to add it by default anyway). The values of this key is describe in table 1.

1.1.2 Task 1: Embedding a file

Embedding an existing file is in most cases quite straightforward. This module offers commands, but it can also be done with the basic commands from the `l3pdf` module `\pdf_object_unnamed_write:nn` or `\pdf_object_new:n`/`\pdf_object_write:nnn` or primitive commands to create objects. The object number should be stored for the reference in the `/Filespec` dictionary.

Table 1: Values of the `/AFRelationship` key

Source	shall be used if this file specification is the original source material for the associated content.
Data	shall be used if this file specification represents information used to derive a visual presentation – such as for a table or a graph.
Alternative	shall be used if this file specification is an alternative representation of content, for example audio.
Supplement	shall be used if this file specification represents a supplemental representation of the original source or data that may be more easily consumable (e.g., A MathML version of an equation).
EncryptedPayload	shall be used if this file specification is an encrypted payload document that should be displayed to the user if the PDF processor has the cryptographic filter needed to decrypt the document.
FormData	shall be used if this file specification is the data associated with the AcroForm (see 12.7.3, "Interactive form dictionary") of this PDF.
Schema	shall be used if this file specification is a schema definition for the associated object (e.g. an XML schema associated with a metadata stream).
Unspecified	(default value) shall be used when the relationship is not known or cannot be described using one of the other values.
Other names	Second-class names (see Annex E, "(normative) PDF Name Registry") should be used to represent other types of relationships.

```
\pdf_object_unnamed_write:ne {fstream}
  {
    {
      /Type /EmbeddedFile
      /Subtype /application\c_hash_str2Fpostscript
      /Params
        <<
          /ModDate  ~ (\file_timestamp:n{example-image.eps})
          /Size     ~  \file_size:n {example-image.eps}
          /CheckSum ~ <\file_mdfive_hash:n {example-image.eps}>
        >>
    }
    {example-image.eps}
  }
\tl_set:Ne \l_my_fileobj_tl {\pdf_object_ref_last:}
```

- The /Params dictionary is not always required, but the commands of these module will prefill them as shown in the examples. A /CreationDate entry has to be added explicitly as there is no sensible way to retrieve this automatically.

- The mimetype (in the /Subtype) should be properly escaped. So for example

  `\pdfdict_put:nne{l_pdffile}{Subtype}{/application\string#2Fx-tex}`

  But while it is possible to set the subtype like this, it is normally better to rely on the file extension and to let the code autodetect the subtype. This module contains a property list with maps a number of file extensions to mimetypes and the commands try to detect and fill the mimetype automatically.

- The dictionary can contain additional keys (/Filter, /DecodeParms), see the pdf reference.

1.1.3 Task 2: Creating the /Filespec dictionary

The /Filespec dictionary is a simple dictionary object, and can also be created in various ways. If it refers to an embedded file it should reference it in the /EF key.

1.1.4 Task 3: Referencing the /Filespec dictionary

Using the dictionary reference in annotations and /AF keys is unproblematic.
But to add it to the /EmbeddedFiles name tree so that it appears in the attachment panel requires special care: This name tree is a global resource and uncoordinated access can lead to clashes and files that are not visible or inaccessible. The access here is managed by the l3pdfmanagement module:

`\pdfmanagement_add:nne{Catalog/Names}{EmbeddedFiles}{⟨objref⟩}`

Table 2: Preset values in the file dictionaries

dictionary	key	value
l_pdffile	Type	/EmbeddedFile
l_pdffile/Params	Size	`\file_size:n{\l_pdffile_source_name_str}`
l_pdffile/Params	ModDate	`(\file_timestamp:n {\l_pdffile_source_name_str})`
l_pdffile/Params	CheckSum	`<\file_mdfive_hash:n{\l_pdffile_source_name_str}>`
l_pdffile/streamParams		a /ModDate entry with year/month/date (used with `\pdffile_embed_stream:nnn`)
l_pdffile/Filespec	Type	/Filespec
l_pdffile/Filespec	AFRelationship	Unspecified

1.2 Commands and tools of these module

`file`
`file/Params`
`file/streamParams`
`file/Filespec`

The module predefines and uses a number of local dictionaries for the components of the stream and the **/Filespec** object. These dictionaries are then used by the `\pdffile_-embed_XX`. The content of these dictionaries can be changed by users with the commands from the l3pdfdict module, but it should be done only locally to avoid side effects on uses by other packages/commands.

The preset values are of these dictionaries are shown in table 2.

`\pdffile_embed_file:nnn` `\pdffile_embed_file:nnn {⟨source filename⟩} {⟨target filename⟩} {⟨object name ⟩}`

This commands embeds the file ⟨*source filename*⟩ in the PDF, and creates a **/Filespec** dictionary object named ⟨*object name*⟩. The object name must be unique, it should start with the module name, so e.g. `module/name`. The command uses the content of the local dictionaries `l_pdffile`, `l_pdffile/Params` and `l_pdffile/Filespec` to setup the dictionary entries of the stream object and the **/Filespec** dictionary. The **/F** and **/UF** entry are filled with ⟨*target filename*⟩.

It is an error if both ⟨*target filename*⟩ and ⟨*source filename*⟩ are empty.

If ⟨*target filename*⟩ is empty ⟨*source filename*⟩ is used instead.

If ⟨*source filename*⟩ is empty, only a **/Filespec** dictionary is created.

If the `l_pdffile` dictionary doesn't contain a Subtype entry with the mimetype, the command tries to guess it from the file extension of ⟨*source filename*⟩. Unknown file extensions can be added (or known extension be changed) by adding to or changing the value in the property `\g_pdffile_mimetypes_prop`, see below.

When using `dvips` and `pstopdf` the actual embedding is done by `pstopdf`. `pstopdf` will embed files only if used with the option `-dNOSAFER` and will not be able to use files which are found with `kpathsea`.

⟨*target filename*⟩ doesn't need to be a file name with an extension, but it is recommended as security settings in the pdf viewer can restrict access to known file types.

`\pdffile_embed_stream:nnn`	`\pdffile_embed_stream:nnn {`⟨*content*⟩`} {`⟨*target filename*⟩`} {`⟨*object name*⟩`}`
`\pdffile_embed_stream:nnN`	`\pdffile_embed_stream:nnN {`⟨*content*⟩`} {`⟨*target filename*⟩`} {`⟨*tl var*⟩`}`

This commands embeds the ⟨*content*⟩ in the PDF in a stream objects and creates either a /Filespec dictionary object named ⟨*object name*⟩, or stores the object reference (what you would get with `\pdf_object_ref:n`) in ⟨*tl var*⟩. ⟨*content*⟩ is wrapped in a `\exp_-not:n`. The object name must be unique. The command uses the content of the local dictionaries `l_pdffile`, `l_pdffile/streamParams` and `l_pdffile/Filespec` to setup the dictionary entries of the stream object and the /Filespec dictionary. The /F and /UF entry are filled with ⟨*target filename*⟩. If ⟨*target filename*⟩ is empty the fix name `stream.txt` is used instead.

If the `l_pdffile` dictionary doesn't contain a Subtype entry with the mimetype, the command tries to guess it from the file extension of ⟨*target filename*⟩.

⟨*target filename*⟩ doesn't need to be a file name with an extension, but it is recommended as security settings in the pdf viewer can restrict access to known file types.

The stream should not be too long, at least PS imposes a size limit for strings.

`\pdffile_filespec:nnn`	`\ pdffile_filespec:nnn {`⟨*object name*⟩`}{`⟨*file name*⟩`}`stream object reference
`\pdffile_filespec:nne`	

The previous commands are fine if stream and filespec dictionary can be created together. But there are cases where the `filespec` dictionary should be referenced when the stream object doesn't exist yet. For example in the `AF` key of a structure at the begin of an environment where the stream is created from the body.

This command allows to write a filespec dictionary alone and reference a previously created stream.

```
\pdf_object_new:n {module/filespec/A} % a new filespec object
\pdf_object_ref:n {module/filespec/A} % reference it somewhere, e.g. in AF
% now write the stream
\pdf_object_unnamed_write:nn { stream }{ {...}{content} }
% and fill and write the filespec dictionary:
\pdffile_filespec:nnn {module/filespec/A}{A.xml}{\pdf_object_ref_last:}
```

`\g_pdffile_mimetypes_prop`	This property contains a list of extensions and their mimetypes. Values can be added or changed with the standard commands:

`\prop_gput:Nnn \g_pdffile_mimetypes_prop {.abc}{text/plain}`

The extension should start with a period, the mimetype should be given as plain text (it will be escaped internally). Extensions with two periods are not supported.

\g_pdffile_embed_pdfa_int
\g_pdffile_embed_nonpdfa_int

These two integers hold the number of embedded files in PDF/A format and non-PDF/A format and can be used for a rough test for the requirements in l3pdfmeta `no_embed_content` (both should be zero) and `only_pdfa_embed_content` (the second should be zero). The commands `\pdffile_embed_stream:...` and `\pdffile_embed_file:...` increase the integers. As the code can currently not detect if an embedded file follows a PDF/A standard it simply goes by the extension: files embedded as `.pdf` increase the first integer.

`\pdffile_filespec:nnn` does *not* increase the integers, if this command is used it lies in the responsability of the author to adjust the integers.

The integers are public so that user can query and adjust the values, e.g. in tests for a standard compliancy.

\l_pdffile_source_name_str This variable is set at the begin of `\pdffile_embed_file:nnn`. It can be (and is) used in the file dictionaries, see table 2 for examples.

\g_pdffile_embed_prop This property holds a list of embedded files. It is used by the following show command. The keys are the object names, the argument holds a key word, the source file name and the target file name. It stores a file only if the boolean `\l_pdffile_embed_show_bool` is true when the file is embedded.

\l_pdffile_embed_show_bool This boolean is used to decided if a file should be stored in the property or not. As storing can be slow if there are many files, it is false by default.

\pdffile_embed_show: This shows the stored embedded files with their source and target name.

1.3 Example

```
\group_begin:
%set the relationship:
\pdfdict_put:nnn  {l_pdffile/Filespec} {AFRelationship}{/Source}
%set the description key. The text must first be converted:
\pdf_string_from_unicode:nnN {utf16/string}
   {this~is~an~odd~description~with~öäü}
   \l_tmpa_str
\pdfdict_put:nne  {l_pdffile/Filespec} {Desc}{\l_tmpa_str}
%embeds testinput.txt and calls it grüße.txt
\pdffile_embed_file:nnn {testinput.txt}{grüße.txt}{mymodule/example1}
%reference it in the panel
\pdfmanagement_add:nne
 {Catalog/Names}
 {EmbeddedFiles}
  {\pdf_object_ref:n{mymodule/example1}}
\group_end:
```

2 **l3pdffile** implementation

```
1 ⟨∗header⟩
2 \ProvidesExplPackage{l3pdffile}{2025-08-05}{0.96v}
3   {embedding and referencing files in PDF---LaTeX PDF management bundle}
4 \RequirePackage{l3pdftools}  %temporarily!!
5 ⟨/header⟩

6 ⟨∗package⟩
7 ⟨@@=pdffile⟩
8 \cs_new_protected:Npn \__pdffile_filename_convert_to_print:nN #1 #2
9   {\pdf_string_from_unicode:nnN {utf16/hex}{#1}{#2}}
```

2.1 Messages

```
10 \msg_new:nnn { pdffile } { file-not-found }
11   {
12     File~'\tl_to_str:n{#1}'~not~found
13   }
14
15 \msg_new:nnn { pdffile } { mimetype-missing }
16   {
17     Mime~type~not~set~for~file~'\tl_to_str:n{#1}'
18   }
19
20 \msg_new:nnn { pdffile } { target-name-missing }
21   {
22     a~target~name~for~the~/Filespec~dictionary~is~missing.
23   }
24
25 \msg_new:nnn { pdffile } { object-exists }
26   {
27     object~name~'#1'~is~already~used.
28   }
29
30 \msg_new:nnn  { pdffile } { show-files }
31   {
32     The~following~files~have~been~embedded\\
33      #1
34   }
```

2.2 Variables

`\l__pdffile_tmpa_tl`
`\l__pdffile_tmpb_tl`
`\g__pdffile_tmpa_tl`
`\l__pdffile_tmpa_str`
`\l__pdffile_tmpb_str`
`\l__pdffile_ext_str`
`\l__pdffile_automimetype_tl`
`\l__pdffile_embed_ref_tl`

temporary variables: generic, for extension, subtype, to store the ref.

(*End of definition for* `\l__pdffile_tmpa_tl` *and others.*)

```
35 \tl_new:N  \l__pdffile_tmpa_tl
36 \tl_new:N  \l__pdffile_tmpb_tl
37 \tl_new:N  \g__pdffile_tmpa_tl
38 \str_new:N \l__pdffile_tmpa_str
39 \str_new:N \l__pdffile_tmpb_str
40 \str_new:N \l__pdffile_ext_str
41 \tl_new:N  \l__pdffile_automimetype_tl
42 \tl_new:N  \l__pdffile_embed_ref_tl
```

\g_pdffile_mimetypes_prop This variable holds common mimetypes. The key is an extension with (one) period, the value the description, e.g. `text/csv`.

(*End of definition for* `\g_pdffile_mimetypes_prop`. *This variable is documented on page 6.*)

```
43 \prop_new:N \g_pdffile_mimetypes_prop
44 \prop_gset_from_keyval:Nn \g_pdffile_mimetypes_prop
45   {
46     ,.css = text/css
47     ,.csv = text/csv
48     ,.html= text/html
49     ,.dtx = text/plain %or application/x-tex, not in iana.org list
50     ,.eps = application/postscript
51     ,.jpg = image/jpeg
52     ,.mp4 = video/mp4
53     ,.pdf = application/pdf
54     ,.png = image/png
55     ,.tex = application/x-tex %not in iana.org list but probably better
56     ,.txt = text/plain
57     ,.sty = text/plain
58     ,.xml = application/xml
59   }
```

\g_pdffile_embed_pdfa_int
\g_pdffile_embed_nonpdfa_int These two integers hold the number of embedded files in PDF/A format and non-PDF/A format and can be used for a rough test for the requirements in l3pdfmeta `no_embed_content` (both should be zero) and `only_pdfa_embed_content` (the second should be zero). The commands `\pdffile_embed_stream:...` and `\pdffile_embed_file:...` increase the integers. As the code can currently not detect if an embedded file follows a PDF/A standard it simply goes by the extension: files embedded as `.pdf` increase the first integer.

`\pdffile_filespec:nnn` does *not* increase the integers, if this command is used it lies in the responsability of the author to adjust the integers.

The integers are public so that user can query and adjust the values, e.g. in tests for a standard compliancy.

```
60 \int_new:N\g_pdffile_embed_pdfa_int
61 \int_new:N\g_pdffile_embed_nonpdfa_int
```

(*End of definition for* `\g_pdffile_embed_pdfa_int` *and* `\g_pdffile_embed_nonpdfa_int`. *These variables are documented on page 7.*)

\l_pdffile_source_name_str `\l_pdffile_source_name_str` will be set at the begin of the command and contains the full file name and can be used e.g. with `\file_timestamp:n`.

(*End of definition for* `\l_pdffile_source_name_str`. *This variable is documented on page 7.*)

```
62 \str_new:N  \l_pdffile_source_name_str
```

Here we define and setup the local dictionaries. We add a ModDate to ensure that there is an entry if associated files are used.

```
63 \pdfdict_new:n { l_pdffile }
64 \pdfdict_put:nnn { l_pdffile }{Type}{/EmbeddedFile}
65 \pdfdict_new:n { l_pdffile/Params }
66 \pdfdict_put:nnn { l_pdffile/Params }
```

9

```
67    {ModDate}  { (\file_timestamp:n { \l_pdffile_source_name_str }) }
68  \pdfdict_put:nnn { l_pdffile/Params }
69    {Size}     { \file_size:n { \l_pdffile_source_name_str } }
70  \pdfdict_put:nnn { l_pdffile/Params }
71    {CheckSum} { <\file_mdfive_hash:n { \l_pdffile_source_name_str }> }
72  \pdfdict_new:n { l_pdffile/streamParams }
73  \pdfdict_put:nnn { l_pdffile/streamParams }
74    {ModDate}  {
75                    (
76                      D:\int_use:N\c_sys_year_int
77                       \int_compare:nNnT{\c_sys_month_int}<{10}{0}
78                       \int_use:N\c_sys_month_int
79                       \int_compare:nNnT{\c_sys_day_int}<{10}{0}
80                       \int_use:N\c_sys_day_int
81                    )
82                 }
83  \pdfdict_new:n { l_pdffile/Filespec }
84  \pdfdict_put:nnn { l_pdffile/Filespec }
85    {Type} { /Filespec }
86  \pdfdict_put:nnn { l_pdffile/Filespec }
87    {AFRelationship} { /Unspecified }
88
```

\g_pdffile_embed_prop
\l_pdffile_embed_show_bool

we record here the relation

⟨*object name*⟩ ⇒ {⟨*file/stream or empty*⟩}{⟨*sourcename*⟩}{⟨*targetname*⟩} if the boolean is true

```
89  \prop_new_linked:N \g_pdffile_embed_prop
90  \bool_new:N \l_pdffile_embed_show_bool
```

(*End of definition for* \g_pdffile_embed_prop *and* \l_pdffile_embed_show_bool. *These variables are documented on page 7.*)

\pdffile_embed_show:

```
91  \cs_new_protected:Npn \pdffile_embed_show:
92    {
93      \msg_show:nne
94        {pdffile}{show-files}
95        {
96          \prop_map_function:NN {\g_pdffile_embed_prop} \msg_show_item:nn
97        }
98    }
```

(*End of definition for* \pdffile_embed_show:. *This function is documented on page 7.*)

\pdffile_embed_file:nnn
\pdffile_embed_stream:nnn
\pdffile_embed_stream:nnN

At first a command to set the mimetype. It either uses the current value in the file dictionary, or tries to guess it from the extension.

__pdffile_mimetype_set:nNN
__pdffile_mimetype_set:VNN
__pdffile_fstream_write:nN
__pdffile_fstream_write:VN
__pdffile_stream_write:nN
__pdffile_stream_write:VN

```
99   %#1 file name,
100  %#2 tl to return the (printed) value for the guessed mimetype
101  %#3 tl to return the file extension (that is a string)
102  \cs_new_protected:Npn \__pdffile_mimetype_set:nNN #1 #2 #3
```

10

```
103     {
104       \file_parse_full_name:nNNN
105           {#1}
106           \l__pdffile_tmpa_str %unused
107           \l__pdffile_tmpb_str %unused
108           \l__pdffile_ext_str
109       %check if Subtype has been set
110       \pdfdict_get:nnN { l_pdffile}{Subtype}\l__pdffile_tmpa_tl
111       %if not look up in the prop:
112       \quark_if_no_value:NT \l__pdffile_tmpa_tl
113         {
114           \prop_get:NVNTF
115             \g_pdffile_mimetypes_prop
116             \l__pdffile_ext_str
117             \l__pdffile_tmpb_tl
118             {
119               \tl_set:Ne #2 {/Subtype~\pdf_name_from_unicode_e:V \l__pdffile_tmpb_tl}
120             }
121             {
122               \msg_warning:nne { pdffile }{ mimetype-missing} {#1}
123               \tl_clear:N #2
124             }
125         }
126       \tl_set_eq:NN #3 \l__pdffile_ext_str
127     }
128
129 \cs_generate_variant:Nn \__pdffile_mimetype_set:nNN {VNN}
130
131 % #1 tl containing a file extension
132 \cs_new_protected:Npn \__pdffile_count_embed:N #1
133   {
134     \str_if_eq:VnTF #1 {.pdf}
135       {\int_gincr:N \g_pdffile_embed_pdfa_int }
136       {\int_gincr:N \g_pdffile_embed_nonpdfa_int }
137   }
138
139 %#1 file name,
140 %#2 tl, should be empty or contain /Subtype /mimetype
141 %   e.g. result from \__pdffile_mimetype_set:nNN
142 \cs_new_protected:Npn \__pdffile_fstream_write:nN #1 #2
143   {
144     \pdf_object_unnamed_write:ne { fstream }
145       {
146         {
147           #2
148           \pdfdict_use:n { l_pdffile}
149           \pdfdict_if_empty:nF { l_pdffile/Params}
150             {
151               /Params
152                 <<
153                   \pdfdict_use:n { l_pdffile/Params}
154                 >>
155             }
156         }
```

11

```
157          { #1 }
158        }
159      \tl_clear:N \l__pdffile_automimetype_tl
160    }
161
162 \cs_generate_variant:Nn \__pdffile_fstream_write:nN {VN}
163
164 %#1 file content
165 %#2 tl, should be empty or contain /Subtype /mimtype
166 %   e.g. result from \__pdffile_mimetype_set:nNN
167 \cs_new_protected:Npn \__pdffile_stream_write:nN #1 #2
168  {
169    \pdf_object_unnamed_write:ne { stream }
170      {
171        {
172          #2
173          \pdfdict_use:n { l_pdffile}
174          \pdfdict_if_empty:nF { l_pdffile/streamParams}
175            {
176              /Params
177                <<
178                  \pdfdict_use:n { l_pdffile/streamParams}
179                >>
180            }
181        }
182        { \exp_not:n { #1 } }
183      }
184    \tl_clear:N \l__pdffile_automimetype_tl
185  }
186
187 \cs_generate_variant:Nn \__pdffile_stream_write:nN {VN}
188
189 %#1 symbolic name of dict object
190 %#2 target file name,
191 %#3 object ref of the file stream.
192 \cs_new_protected:Npn \__pdffile_filespec_write:nnn #1 #2 #3
193  {
194    \tl_if_blank:nTF { #2 }
195      {
196        \msg_error:nn {pdffile}{target-name-missing}
197      }
198      {
199        \group_begin:
200          \pdf_string_from_unicode:nnN {utf8/string}{#2}\l__pdffile_tmpa_str
201          \pdfdict_put:nne {l_pdffile/Filespec}{F} { \l__pdffile_tmpa_str }
202          \__pdffile_filename_convert_to_print:nN  { #2 } \l__pdffile_tmpa_str
203          \pdfdict_put:nne {l_pdffile/Filespec}{UF}{ \l__pdffile_tmpa_str }
204          \pdf_object_write:nne { #1 } { dict }
205            {
206              \pdfdict_use:n { l_pdffile/Filespec}
207              \tl_if_empty:nF { #3 }
208                {
209                  /EF <</F~#3 /UF~#3>>
210                }
```

```
211              }
212            \group_end:
213          }
214      }
215
216  %#1 target file name #2 object ref of file stream #3 reference of object
217  \cs_new_protected:Npn \__pdffile_filespec_write:nnN #1 #2 #3
218    {
219      \tl_if_blank:nTF { #1 }
220        {
221          \msg_error:nn {pdffile}{target-name-missing}
222        }
223        {
224          \group_begin:
225            \pdf_string_from_unicode:nnN {utf8/string}{#1}\l__pdffile_tmpa_str
226            \pdfdict_put:nne {l_pdffile/Filespec}{F} { \l__pdffile_tmpa_str }
227            \__pdffile_filename_convert_to_print:nN  { #1 } \l__pdffile_tmpa_str
228            \pdfdict_put:nne {l_pdffile/Filespec}{UF}{ \l__pdffile_tmpa_str }
229            \pdf_object_unnamed_write:ne {dict}
230              {
231                \pdfdict_use:n { l_pdffile/Filespec}
232                \tl_if_empty:nF { #2 }
233                  {
234                    /EF <</F~#2 /UF~#2>>
235                  }
236              }
237            \tl_gset:Ne\g__pdffile_tmpa_tl{\pdf_object_ref_last:}
238          \group_end:
239          \tl_set_eq:NN#3\g__pdffile_tmpa_tl
240        }
241    }
242
243  \cs_set_eq:NN \pdffile_filespec:nnn \__pdffile_filespec_write:nnn
244  \cs_generate_variant:Nn \pdffile_filespec:nnn {nne,nnx}
245  %#1 {source filename}
246  %#2 {target filename}
247  %#3 { filespec object name } (will internally get a prefix! ??)
248  \cs_new_protected:Npn \pdffile_embed_file:nnn #1 #2 #3
249    { %                  if #1 empty => only filespec
250      %                  if #2 empty => = #1
251      \pdf_object_if_exist:nTF { #3 }
252        {
253          \msg_error:nnn { pdffile }{ object-exists } { #3 }
254        }
255        {
256          \tl_if_blank:nTF { #1 }
257            {
258              \tl_set:Nn \l__pdffile_embed_ref_tl {}
259            }
260            {
261              \file_get_full_name:nNTF {#1} \l_pdffile_source_name_str
262                {
263                  \__pdffile_mimetype_set:VNN
264                    \l_pdffile_source_name_str
```

13

```
265            \l__pdffile_automimetype_tl
266            \l__pdffile_tmpa_tl
267          \__pdffile_count_embed:N \l__pdffile_tmpa_tl
268          \__pdffile_fstream_write:VN
269            \l_pdffile_source_name_str
270            \l__pdffile_automimetype_tl
271          \tl_set:Ne \l__pdffile_embed_ref_tl { \pdf_object_ref_last: }
272        }
273        {
274          \msg_error:nnn { pdffile }{ file-not-found }{ #1 }
275        }

277      }
278    \bool_if:NT\l_pdffile_embed_show_bool
279     {
280       \prop_gput:Nne
281         \g_pdffile_embed_prop
282         { #3 }
283         {
284           { \tl_if_blank:nTF { #1 } {filespec}{file} }
285           {\l_pdffile_source_name_str}
286           {
287             \tl_if_blank:nTF { #2 }
288               { \l_pdffile_source_name_str }
289               { \tl_to_str:n{#2}}
290           }
291         }
292       }
293    \tl_if_blank:nTF { #2 }
294      {
295        \pdf_object_new:n    { #3 }
296        \exp_args:Nnne
297          \__pdffile_filespec_write:nnn
298          %#1 dict, #2 target file name, #3 object ref
299          { #3 }
300          { #1 }
301          {\l__pdffile_embed_ref_tl}
302      }
303      {
304        \pdf_object_new:n    { #3 }
305        \exp_args:Nnne
306          \__pdffile_filespec_write:nnn
307          %#1 dict, #2 target file name, #3 object ref
308          { #3 }
309          { #2 }
310          {\l__pdffile_embed_ref_tl}
311      }
312    }
313  }


316 %#1{stream content}
317 %#2{target filename}
318 %#3{file object name }
```

```
319 \cs_new_protected:Npn \pdffile_embed_stream:nnn #1 #2 #3
320   {
321     %                   if #2 empty => error
322     \pdf_object_if_exist:nTF { #3 }
323       {
324         \msg_error:nnn { pdffile }{ object-exists } { #3 }
325       }
326       {
327         \bool_if:NT\l_pdffile_embed_show_bool
328         {
329           \prop_gput:Nne
330             \g_pdffile_embed_prop
331             { #3 }
332             {{stream}{}{\tl_if_blank:nTF {#2}{stream.txt}{\exp_not:n{#2}}}}}
333         }
334         \tl_if_blank:nTF {#2}
335          { \__pdffile_mimetype_set:nNN {stream.txt}\l__pdffile_automimetype_tl \l__pdffile_t
336          { \__pdffile_mimetype_set:nNN { #2 } \l__pdffile_automimetype_tl \l__pdffile_tmpa_t
337         \__pdffile_count_embed:N \l__pdffile_tmpa_tl
338         \__pdffile_stream_write:nN
339           { #1 }
340           \l__pdffile_automimetype_tl
341         \tl_set:Ne \l__pdffile_embed_ref_tl { \pdf_object_ref_last: }
342         \pdf_object_new:n    { #3 }
343         \exp_args:Nnee
344           \__pdffile_filespec_write:nnn
345             %#1 dict, #2 target file name, #3 object ref
346             { #3 }
347             { \tl_if_blank:nTF {#2}{stream.txt}{\exp_not:n{#2}} }
348             {\l__pdffile_embed_ref_tl}
349       }
350   }
351
352 \cs_new_protected:Npn \pdffile_embed_stream:nnN #1 #2 #3
353   {
354     \tl_if_blank:nTF {#2}
355      { \__pdffile_mimetype_set:nNN {stream.txt}\l__pdffile_automimetype_tl \l__pdffile_tmpa_
356      { \__pdffile_mimetype_set:nNN { #2 } \l__pdffile_automimetype_tl \l__pdffile_tmpa_tl }
357     \__pdffile_count_embed:N\l__pdffile_tmpa_tl
358     \__pdffile_stream_write:nN
359       { #1 }
360       \l__pdffile_automimetype_tl
361     \tl_set:Ne \l__pdffile_embed_ref_tl { \pdf_object_ref_last: }
362     \exp_args:Nee
363       \__pdffile_filespec_write:nnN
364         %#1 target file name, #2 object ref of stream, #3 object ref of filespec
365         { \tl_if_blank:nTF {#2}{stream.txt}{\exp_not:n{#2}} }
366         {\l__pdffile_embed_ref_tl}
367         #3
368     \bool_if:NT \l_pdffile_embed_show_bool
369       {
370         \prop_gput:Nee
371           \g_pdffile_embed_prop
372           { #3 }
```

```
373                    {{stream}{}{\tl_if_blank:nTF {#2}{stream.txt}{\exp_not:n{#2}}}}}
374            }
375       }
376
377
```

(*End of definition for* `\pdffile_embed_file:nnn` *and others. These functions are documented on page 5.*)

```
378  ⟨/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.

16