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

The LaTeX Project*

Version 0.95g, released 2021-07-21

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:nn`/`\pdf_object_write:nn` 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:nx {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:Nx \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. 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:nnx{Catalog/Names}{EmbeddedFiles}{`⟨*objref*⟩`}`

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.

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

`\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 ⟩}`

This commands embeds the ⟨*content*⟩ in the PDF in a stream objects and creates a `/Filespec` dictionary object named ⟨*object name*⟩. ⟨*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:nnx

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

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:nn{module/filespec/A}{dict} % a new filespec object
\pdf_object_ref:n {module/filespec/A}       % a reference
\pdf_object_unnamed_write:nn { stream }{ {...}{content} } %writing the stream
% filling and writing 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.

\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.

\pdffile_embed_show:

This shows the 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:nnx  {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:nnx
 {Catalog/Names}
```

```
 {EmbeddedFiles}
  {\pdf_object_ref:n{mymodule/example1}}
\group_end:
```

2 l3pdffile implementation

```
⟨*header⟩
\ProvidesExplPackage{l3pdffile}{2021-07-21}{0.95g}
  {embedding and referencing files in PDF---LaTeX PDF management testphase bundle}
\RequirePackage{l3pdftools}  %temporarly!!
⟨/header⟩
⟨*package⟩
⟨@@=pdffile⟩
\cs_new_protected:Npn \__pdffile_filename_convert_to_print:nN #1 #2
  {\pdf_string_from_unicode:nnN {utf16/hex}{#1}{#2}}
```

2.1 Messages

```
\msg_new:nnn { pdffile } { file-not-found }
  {
    File~'\tl_to_str:n{#1}'~not~found
  }

\msg_new:nnn { pdffile } { mimetype-missing }
  {
    Mime~type~not~set~for~file~'\tl_to_str:n{#1}'
  }

\msg_new:nnn { pdffile } { target-name-missing }
  {
    a~target~name~for~the~/Filespec~dictionary~is~missing.
  }

\msg_new:nnn { pdffile } { object-exists }
  {
    object~name~'#1'~is~already~used.
  }

\msg_new:nnn  { pdffile } { show-files }
  {
    The~following~files~have~been~embedded\\
     #1
  }
```

\l__pdffile_tmpa_tl
\l__pdffile_tmpb_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 definition for* `\l__pdffile_tmpa_tl` *and others.*)

```
\tl_new:N  \l__pdffile_tmpa_tl
\tl_new:N  \l__pdffile_tmpb_tl
\str_new:N \l__pdffile_tmpa_str
\str_new:N \l__pdffile_tmpb_str
\str_new:N \l__pdffile_ext_str
\tl_new:N  \l__pdffile_automimetype_tl
\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 definition for* `\g_pdffile_mimetypes_prop`. *This variable is documented on page* 6.)

```
\prop_new:N \g_pdffile_mimetypes_prop
\prop_set_from_keyval:Nn \g_pdffile_mimetypes_prop
  {
    ,.csv = text/csv
    ,.html= text/html
    ,.dtx = text/plain %or application/x-tex, not in iana.org list
    ,.eps = application/postscript
    ,.jpg = image/jpeg
    ,.mp4 = video/mp4
    ,.pdf = application/pdf
    ,.png = image/png
    ,.tex = text/plain %or application/x-tex, not in iana.org list
    ,.txt = text/plain
    ,.sty = text/plain
    ,.xml = text/xml
  }
```

\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 definition for* `\l_pdffile_source_name_str`. *This variable is documented on page* 6.)

```
\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.

```
\pdfdict_new:n { l_pdffile }
\pdfdict_put:nnn { l_pdffile }{Type}{/EmbeddedFile}
\pdfdict_new:n { l_pdffile/Params }
\pdfdict_put:nnn { l_pdffile/Params }
  {ModDate}  { (\file_timestamp:n { \l_pdffile_source_name_str }) }
\pdfdict_put:nnn { l_pdffile/Params }
  {Size}     { \file_size:n { \l_pdffile_source_name_str } }
\pdfdict_put:nnn { l_pdffile/Params }
  {CheckSum} { (\file_mdfive_hash:n { \l_pdffile_source_name_str }) }
\pdfdict_new:n { l_pdffile/streamParams }
\pdfdict_put:nnn { l_pdffile/streamParams }
  {ModDate}  {
               (
                 D:\int_use:N\c_sys_year_int
                  \int_compare:nNnT{\c_sys_month_int}<{10}{0}
                  \int_use:N\c_sys_month_int
                  \int_compare:nNnT{\c_sys_day_int}<{10}{0}
                  \int_use:N\c_sys_day_int
               )
             }
\pdfdict_new:n { l_pdffile/Filespec }
\pdfdict_put:nnn { l_pdffile/Filespec }
  {Type} { /Filespec }
\pdfdict_put:nnn { l_pdffile/Filespec }
  {AFRelationship} { /Unspecified }

```

\g_pdffile_embed_prop

we record here the relation
⟨*object name*⟩ ⇒ {⟨*file/stream or empty*⟩}{⟨*sourcename*⟩}{⟨*targetname*⟩}

```
\prop_new:N \g_pdffile_embed_prop
```

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

\pdffile_embed_show:

```
\cs_new_protected:Npn \pdffile_embed_show:
  {
    \msg_show:nnx
     {pdffile}{show-files}
     {
       \prop_map_function:NN {\g_pdffile_embed_prop} \msg_show_item:nn
     }
  }
```

(*End definition for* `\pdffile_embed_show:`. *This function is documented on page* 6.)

\pdffile_embed_file: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:nN
__pdffile_mimetype_set:VN
__pdffile_fstream_write:nN
__pdffile_fstream_write:VN
__pdffile_stream_write:nN
__pdffile_stream_write:VN

```
%#1 file name,
%#2 tl to return the (printed) value for the guessed mimetype
\cs_new_protected:Npn \__pdffile_mimetype_set:nN #1 #2
  {
    \file_parse_full_name:nNNN
          {#1}
          \l__pdffile_tmpa_str %unused
          \l__pdffile_tmpb_str %unused
          \l__pdffile_ext_str
        %check if Subtype has been set
        \pdfdict_get:nnN { l_pdffile}{Subtype}\l__pdffile_tmpa_tl
        %if not look up in the prop:
        \quark_if_no_value:NT \l__pdffile_tmpa_tl
          {
            \prop_get:NVNTF
              \g_pdffile_mimetypes_prop
              \l__pdffile_ext_str
              \l__pdffile_tmpb_tl
              {
                \tl_set:Nx #2 {/Subtype~\pdf_name_from_unicode_e:V \l__pdffile_tmpb_tl}
              }
              {
                \msg_warning:nnx { pdffile }{ mimetype-missing} {#1}
                \tl_clear:N #2
              }
          }
    }

\cs_generate_variant:Nn \__pdffile_mimetype_set:nN {VN}

%#1 file name,
%#2 tl, should be empty or contain /Subtype /mimetype
```

```
%   e.g. result from \__pdffile_mimetype_set:NN
\cs_new_protected:Npn \__pdffile_fstream_write:nN #1 #2
  {
    \pdf_object_unnamed_write:nx { fstream }
      {
        {
          #2
          \pdfdict_use:n { l_pdffile}
          \pdfdict_if_empty:nF { l_pdffile/Params}
            {
              /Params
                <<
                  \pdfdict_use:n { l_pdffile/Params}
                >>
            }
        }
        { #1 }
      }
     \tl_clear:N \l__pdffile_automimetype_tl
   }

\cs_generate_variant:Nn \__pdffile_fstream_write:nN {VN}

%#1 file content
%#2 tl, should be empty or contain /Subtype /mimtype
%   e.g. result from \__pdffile_mimetype_set:NN
\cs_new_protected:Npn \__pdffile_stream_write:nN #1 #2
  {
    \pdf_object_unnamed_write:nx { stream }
      {
        {
          #2
          \pdfdict_use:n { l_pdffile}
          \pdfdict_if_empty:nF { l_pdffile/streamParams}
            {
              /Params
                <<
                  \pdfdict_use:n { l_pdffile/streamParams}
                >>
            }
        }
        { \exp_not:n { #1 } }
      }
     \tl_clear:N \l__pdffile_automimetype_tl
   }

\cs_generate_variant:Nn \__pdffile_stream_write:nN {VN}

%#1 symbolic name of dict object
%#2 target file name,
%#3 object ref of the file stream.
\cs_new_protected:Npn \__pdffile_filespec_write:nnn #1 #2 #3
  {
    \tl_if_blank:nT { #2 }
```

```
      {
        \msg_error:nn {pdffile}{target-name-missing}
      }
      {
        \group_begin:
          \__pdffile_filename_convert_to_print:nN  { #2 } \l__pdffile_tmpa_str
          \pdfdict_put:nnx {l_pdffile/Filespec}{F} { \l__pdffile_tmpa_str }
          \pdfdict_put:nnx {l_pdffile/Filespec}{UF}{ \l__pdffile_tmpa_str }
          \pdf_object_write:nx { #1 }
            {
              \pdfdict_use:n { l_pdffile/Filespec}
              \tl_if_empty:nF { #3 }
                {
                  /EF <</F~#3 /UF~#3>>
                }
            }
        \group_end:
      }
  }

\cs_set_eq:NN \pdffile_filespec:nnn \__pdffile_filespec_write:nnn
\cs_generate_variant:Nn \pdffile_filespec:nnn {nnx}
%#1 {source filename}
%#2 {target filename}
%#3 { filespec object name } (will internally get a prefix! ??)
\cs_new_protected:Npn \pdffile_embed_file:nnn #1 #2 #3
  { %               if #1 empty => only filespec
    %               if #2 empty => = #1
    \pdf_object_if_exist:nTF { #3 }
      {
        \msg_error:nnn { pdffile }{ object-exists } { #3 }
      }
      {
        \tl_if_blank:nTF { #1 }
          {
            \tl_set:Nn \l__pdffile_embed_ref_tl {}
          }
          {
            \file_get_full_name:nNTF {#1} \l_pdffile_source_name_str
              {
                \__pdffile_mimetype_set:VN
                  \l_pdffile_source_name_str
                  \l__pdffile_automimetype_tl
                \__pdffile_fstream_write:VN
                  \l_pdffile_source_name_str
                  \l__pdffile_automimetype_tl
                \tl_set:Nx \l__pdffile_embed_ref_tl { \pdf_object_ref_last: }
              }
              {
                \msg_error:nnn { pdffile }{ file-not-found }{ #1 }
              }

           }
        \prop_gput:Nnx
```

```
            \g_pdffile_embed_prop
            { #3 }
            {
              { \tl_if_blank:nTF { #1 } {filespec}{file} }
              {\l_pdffile_source_name_str}
              {
                \tl_if_blank:nTF { #2 }
                  { \l_pdffile_source_name_str }
                  { \tl_to_str:n{#2}}
              }
            }
          \tl_if_blank:nTF { #2 }
            {
              \pdf_object_new:nn   { #3 } {dict}
              \exp_args:Nnnx
                \__pdffile_filespec_write:nnn
                  %#1 dict, #2 target file name, #3 object ref
                  { #3 }
                  { #1 }
                  {\l__pdffile_embed_ref_tl}
            }
            {
              \pdf_object_new:nn   { #3 } {dict}
              \exp_args:Nnnx
                \__pdffile_filespec_write:nnn
                  %#1 dict, #2 target file name, #3 object ref
                  { #3 }
                  { #2 }
                  {\l__pdffile_embed_ref_tl}
            }
        }
    }


%#1{stream content}
%#2{target filename}
%#3{file object name }
\cs_new_protected:Npn \pdffile_embed_stream:nnn #1 #2 #3
  {
    %               if #2 empty => error
    \pdf_object_if_exist:nTF { #3 }
      {
         \msg_error:nnn { pdffile }{ object-exists } { #3 }
      }
      {
         \prop_gput:Nnx
           \g_pdffile_embed_prop
           { #3 }
           {{stream}{}{\tl_if_blank:nTF {#2}{stream.txt}{\exp_not:n{#2}}}}
         \tl_if_blank:nTF {#2}
          { \__pdffile_mimetype_set:nN {stream.txt}\l__pdffile_automimetype_tl}
          { \__pdffile_mimetype_set:nN { #2 } \l__pdffile_automimetype_tl }
         \__pdffile_stream_write:nN
           { #1 }
```

```
      \l__pdffile_automimetype_tl
    \tl_set:Nx \l__pdffile_embed_ref_tl { \pdf_object_ref_last: }
    \pdf_object_new:nn   { #3 } {dict}
    \exp_args:Nnxx
      \__pdffile_filespec_write:nnn
        %#1 dict, #2 target file name, #3 object ref
        { #3 }
        { \tl_if_blank:nTF {#2}{stream.txt}{\exp_not:n{#2}} }
        {\l__pdffile_embed_ref_tl}
  }
}
```

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

⟨/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.

Q

R

S

T