The l3pdfoutline module
Commands for PDF bookmarks
LaTeX PDF management bundle

The LaTeX Project*

Version 0.96i, released 2025-10-12

1 l3pdfoutline documentation

1.1 Introduction

This module contains a number of commands to create the PDF outline also called bookmarks.

Attention: There can be only one outline in a PDF and the commands of this module should not be mixed with other code that create outline items, this would mess up the tree!

The outline is a tree build from a set of objects. The root is the *outline dictionary* which is referenced from the catalog with the `/Outline` key. The other objects are called *outline items*.

The dictionaries of the *outline items* contain keys which can be divided in three classes:

"Management" The connection between the root, the items and their children are described with keys `/Parent`, `/Next`, `/Prev`, `/First`, `/Last`. The `/Count` key describes how many subitems are visible (open).

"Attributes" The `/Title` key contains the text shown in the bookmarks, `/C` its color, the flag `/F` the font type (italic and/or bold). There is a `/SE` key which can reference a structure, but its function is unclear (the note in the reference says, that it is not meant for navigation, but does not say what it is for).

"Action" The action that should be executed can be set either with an `/A` or a `/Dest` key. LaTeX typically uses only the first option and adds an action, e.g. for a simple link to a destination something like `<< /S /GoTo /D (section.1) /SD 19 0 R >>` (the `SD` key is used when there are structures).

While in most cases the outline is used with a `GoTo` action and more or less mirrors the table of contents, other actions are possible too, a outline item can e.g. open an external url or another file. The PDF reference lists in total 17 different, theoretically usable actions[1]: GoTo, GoToR, GoToE, GoToDp (PDF 2.0), Launch, Thread,

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

[1] Sound and Movie are deprecated in PDF 2.0

URI, Sound, Movie, Hide, Named, SetOCGState, Rendition, Trans, GoTo3DView, JavaScript, RichMediaExecute.

An action dictionary typically looks like this:

```
/A << /Type /Action %optional
      /S    ... % type name, e.g. /URI or /GoTo
      /Next ... % optional, next action
      ...   ... % more keys
   >>
```

The `/Next` key allows to chain actions. The value can be a reference to a single action, or an array of actions. It depends on the action type which other keys should be used.

From all these action types the bookmark package supports `GoTo` (through the `dest` and `page` keys), `GoToR` (through the `gotor` key together with the `dest` and `page` keys), `Named` (through the `named` key) and `URI` (through the `uri` key). These standard types looks like this:

GoTo link to a named destination e.g. `/S /GoTo /D (section.1) /SD 19 0 R`, this can be handled with `\pdfoutline_goto:nnn`.

GoTo link to a page `/S /GoTo /D [5 0 R /FitH 0]`

Named e.g. `/S/Named/N/FirstPage`, values are FirstPage, NextPage, PrevPage, LastPage.

URI e.g.`/S/URI/URI(https://blub.de)`

GoToR `/S/GoToR/F(blub.pdf)/D[0/Fit]`, `/S/GoToR/F(blub.pdf)/D[3/FitH 0]`, `/S/GoToR/F(blub.pdf)/D(duck)`

All other actions (and also chained actions) are not directly supported but can be implemented by using the `rawaction` key.

Theoretically all needed objects described above could be created manually, but getting all the management keys right wouldn't be trivial. For `/Count` e.g. one would have to track recursively the number of visible subitems. Luckily all backends have dedicated primitives or `\special`'s to create outline items which take care of the management keys. Not quite so lucky is that the backends use different methods to decide if an outline item is a sibling or a child or some great-uncle of the previous item.

The easiest case is the `(x)dvipdfmx` case: The general syntax is `\special{pdf:out [-]number << attr>>}` where the parameter *number* is an integer representing the level of the outline entry. The main problem is that the first item has to start with the level 1 and that one can't skip levels, so one can't simply use a fix number for the various headings but has to keep track which level has been used previously. Nevertheless with this backend it is possible to build the outline directly while compiling the document and one doesn't have to use the `aux` or to delay code until the end of the document.

When using pdfTEX, luaTEX or the `dvips` backend, a count must be given in the primitive which describes the number of direct children of the outline. The backend then collects (recursively) following items until all children are found and then goes up a level again. To get the number of children the code has obviously to look ahead to e.g. count all subsections below the current subsection. hyperref solved this problem by writing everything into the `.out` file and to process it at begin of the next compilation to get the correct count. The bookmark package writes the content and attributes (hex

encoded) into the `.aux`. Additionally it keeps track of the number of children during the compilation and stores it for every bookmark in a command. When the `.aux` file is read in again at the end of the document, it decodes content and attributes again and so can process the outline commands for pdfTeX and luaTeX directly in the first compilation. With the `dvips` backend it creates a `.ps` file which is then read in at the next compilation.

The implementation here doesn't use external files but keeps all bookmarks in memory and outputs them at the end of the document (for the dvi based engines in the shipout/lastpage hook).

1.2 The commands

1.3 Creating an outline item

`\pdfoutline_action:nnn`
`\pdfoutline_action:(neo|nee)`

`\pdfoutline_action:nnn{⟨level⟩}{⟨action⟩}{⟨title⟩}`

This creates an outline. ⟨*level*⟩ is an integer expression and sets the (relative) level: A positive number creates a child of the current outline, 0 creates a sibling, and negative numbers go back. ⟨*action*⟩ should be a list of dictionary keys and its values that can be used within the `/A` key, e.g. the output from `\pdfdict_use:n`[2][3]. ⟨*title*⟩ is the title, it should be correctly escaped and encoded for a use as a literal string (but without the outer parentheses), such a representation can for example be created with `\pdfstringdef`. The command can be used for general outline item, for the special case of a GoTo link to a named destination see the following command.

`\pdfoutline_goto:nnn`
`\pdfoutline_goto:(neo|nee)`

`\pdfoutline_goto:nnn{⟨level⟩}{⟨destination⟩}{⟨title⟩}`

This creates an outline which creates a link to a named destination. While such a GoTo link can also be created with `\pdfoutline_action:nnn` the use of this command is recommended as it will also create a reference to a structure destination if tagging is activated. ⟨*level*⟩ is an integer expression and sets the (relative) level: A positive number creates a child of the current outline, 0 creates a sibling, and negative numbers go back. ⟨*destination*⟩ is a destination name. ⟨*title*⟩ is the title, it should be correctly escaped and encoded for a use as a literal string (but without the outer parentheses), such a representation can for example be created with `\pdfstringdef`.

`\l_pdfoutline_bool` This boolean decides if bookmarks are shown at all.

`\l_pdfoutline_open_bool` This boolean decides if bookmarks are shown "open", so show their children or not.

`\l_pdfoutline_open_int` With this integer you can set the number of visible levels. Bookmarks with a level above the value of this integer are always closed. So if it set to 1, everything is closed and so only the top level bookmarks are shown if its value is 2, the first level will show its children, etc. By default this integer is set to `\c_max_int` and so everything is open.

[2] It can not be an object reference as this doesn't work with most backends.

[3] This is not completely accurate: the dvips backend expects in some cases special names, e.g. a color must be given with `/Color` and not `/C`. Complex actions must therefore be tested and if needed one must add backend abstractions.

`\l_pdfoutline_F_bitset` This bitset is used to set the flag for the font in bookmarks. It knows the two values `italic` and `bold`, so after

```
\bitset_set_true:Nn \l_pdfoutline_F_bitset {italic}
```

italic will be enabled.

`\l_pdfoutline_color_tl`
`\l_pdfoutline_color_model_tl`

`\l_pdfoutline_color_tl` can be empty or hold a color expression to set the color of the outline(s). It then should have either the format `[model]{value}` or be a color expression. For examples: `[rgb]{1,0,.5}` or `red!50!blue`. `\l_pdfoutline_color_model_tl` should be `rgb` or `cmyk`. The first is the normal one, the option to use also `cmyk` is only offered for the case that some PDF/A validator complains.

1.4 References

A user package that uses these commands to create a tree, must be able to retrieve parents and level data.

`\pdfoutline_id_ref_last:` ⋆ `\pdfoutline_id_ref_last:`
`\pdfoutline_level_ref_last:` ⋆ `\pdfoutline_level_ref_last:`

These expandable function give back the id and the level of the last bookmark created.

`\pdfoutline_parent_ref:n` ⋆ `\pdfoutline_parent_ref:n{`⟨*id*⟩`}`

This retrieves the parent of the bookmark with id ⟨*id*⟩. If the bookmark is at the root level the value is 1.

2 l3pdfoutline implementation

2.1 Package declaration

```
⟨@@=pdfoutline⟩
⟨*header⟩
\ProvidesExplPackage{l3pdfoutline}{2025-10-12}{0.95s}
  {PDF outlines}
⟨/header⟩

⟨*package⟩
```

2.2 Public variables

`\l_pdfoutline_bool` The state of the first boolean decides if a bookmark is created at all, this allows to disable
`\l_pdfoutline_open_bool` bookmarks fully or in parts. The second boolean decides if a bookmark is open (so shows its children) or not.

```
\bool_new:N      \l_pdfoutline_bool
\bool_set_true:N \l_pdfoutline_bool
\bool_new:N      \l_pdfoutline_open_bool
\bool_set_true:N \l_pdfoutline_open_bool
```

(*End of definition for* `\l_pdfoutline_bool` *and* `\l_pdfoutline_open_bool`. *These variables are documented on page 3.*)

`\l_pdfoutline_open_int` This is the level up to which a bookmark is opened. By default the maximum integer value is used and everything is open.

```
\int_new:N  \l_pdfoutline_open_int
\int_set:Nn \l_pdfoutline_open_int { \c_max_int }
```

(*End of definition for* `\l_pdfoutline_open_int`. *This variable is documented on page 3.*)

`\l_pdfoutline_F_bitset` Outlines have a /F flag, we provide a public bitset for it.

```
\bitset_new:Nn \l_pdfoutline_F_bitset
  {
    Italic         = 1,
    italic         = 1,
    Bold           = 2,
    bold           = 2
  }
```

(*End of definition for* `\l_pdfoutline_F_bitset`. *This variable is documented on page 4.*)

`\l_pdfoutline_color_tl` `\l_pdfoutline_color_model_tl` The variable for the color expression.

```
\tl_new:N  \l_pdfoutline_color_tl
\tl_new:N  \l_pdfoutline_color_model_tl
\tl_set:Nn \l_pdfoutline_color_model_tl {rgb}
```

(*End of definition for* `\l_pdfoutline_color_tl` *and* `\l_pdfoutline_color_model_tl`. *These variables are documented on page 4.*)

2.3 Data structure for the Count

`\l__pdfoutline_tmpa_tl`

```
\tl_new:N\l__pdfoutline_tmpa_tl
```

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

`\g__pdfoutline_id_int` This integer is used to track the bookmarks. id 1 is the root, the first "real" bookmark has id 2.

```
\int_new:N   \g__pdfoutline_id_int
\int_gincr:N \g__pdfoutline_id_int
```

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

`\g__pdfoutline_current_parent_id_tl` This holds the id of the current parent so that we do not have to retrieve it all the time.

```
\tl_new:N   \g__pdfoutline_current_parent_id_tl
\tl_gset:Nn \g__pdfoutline_current_parent_id_tl {1}
```

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

\g__pdfoutline_current_level_int

This holds the (relative) level. It starts at 1 as dvipdfmx counts this way.

```
\int_new:N   \g__pdfoutline_current_level_int
\int_gincr:N \g__pdfoutline_current_level_int
```

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

We do not assume that we will have ever more than 2000 bookmarks, but we set a block size so that we can make the range dynamic if needed like in the object code.

\c__pdfoutline_block_size_int

Sets the block size used for managing outlines

```
\int_const:Nn \c__pdfoutline_block_size_int { 2000 }
```

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

For some backends we must know for every outline the numbers of direct children, and we must know the id of the parent to be able to go up the tree again. For this we use two intarray

\g__pdfoutline_kid_count_intarray
\g__pdfoutline_parent_id_intarray

The first contains for every id the number of (direct) kids. The second contains the id of the parent.

```
\intarray_new:Nn   \g__pdfoutline_kid_count_intarray { \c__pdfoutline_block_size_int }
\intarray_new:Nn   \g__pdfoutline_parent_id_intarray { \c__pdfoutline_block_size_int }
\intarray_gset:Nnn \g__pdfoutline_parent_id_intarray {1}{1}
```

(*End of definition for* `\g__pdfoutline_kid_count_intarray` *and* `\g__pdfoutline_parent_id_intarray`.)

Unlike bookmark we do not use an external file but keep all content in memory in one large tl var.

\g__pdfoutline_collect_build_tl

```
\tl_new:N \g__pdfoutline_collect_build_tl
```

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

The main commands

\pdfoutline_action:nnn

```
\cs_new_protected:Npn \pdfoutline_action:nnn #1 #2 #3
% #1 level, #2 action, #3 title
  {
    \bool_if:NT \l_pdfoutline_bool
      {
        \int_compare:nNnTF {\g__pdfoutline_id_int}={1}
         {
           \tl_build_gbegin:N \g__pdfoutline_collect_build_tl
           \__pdfoutline_item:nnnn {0}{#2}{#3}{action}
           \tl_gput_right:Nn \g__kernel_pdfmanagement_end_run_code_tl
             {
               \tl_build_gend:N \g__pdfoutline_collect_build_tl
               \tl_use:N \g__pdfoutline_collect_build_tl
             }
```

```
          }
          {
            \__pdfoutline_item:nnnn {#1}{#2}{#3}{action}
          }
        \cs_gset_protected:Npn \pdfoutline_action:nnn ##1 ##2 ##3
          {
            \bool_if:NT \l_pdfoutline_bool
              {\__pdfoutline_item:nnnn {##1}{##2}{##3}{action}}
          }
      }
  }
\cs_generate_variant:Nn \pdfoutline_action:nnn {nee}
```

(*End of definition for* `\pdfoutline_action:nnn`*. This function is documented on page 3.*)

`\pdfoutline_goto:nnn`

```
\cs_new_protected:Npn \pdfoutline_goto:nnn #1 #2 #3
% #1 level, #2 destination, #3 title
  {
   \bool_if:NT \l_pdfoutline_bool
      {
        \int_compare:nNnTF {\g__pdfoutline_id_int}={1}
          {
            \tl_build_gbegin:N \g__pdfoutline_collect_build_tl
            \__pdfoutline_item:nnnn {0}{#2}{#3}{goto}
            \tl_gput_right:Nn \g__kernel_pdfmanagement_end_run_code_tl
              {
                \tl_build_gend:N \g__pdfoutline_collect_build_tl
                \tl_use:N \g__pdfoutline_collect_build_tl
              }
          }
          {
            \__pdfoutline_item:nnnn {#1}{#2}{#3}{goto}
          }
        \cs_gset_protected:Npn \pdfoutline_goto:nnn ##1 ##2 ##3
          {
            \bool_if:NT \l_pdfoutline_bool
              {\__pdfoutline_item:nnnn {##1}{##2}{##3}{goto}}
          }
      }
  }
\cs_generate_variant:Nn \pdfoutline_goto:nnn {nee}
```

(*End of definition for* `\pdfoutline_goto:nnn`*. This function is documented on page 3.*)

`\__pdfoutline_item:nnnn`

```
\cs_new_protected:Npn \__pdfoutline_item:nnnn #1 #2 #3 #4
% #1 level, #2 action, #3 title, #4 keyword action or goto
  {
    \int_gincr:N \g__pdfoutline_id_int
```

Follow up bookmarks. At first siblings. The parent doesn't change.

```
    \int_compare:nNnTF {#1}={0}
```

```
      {
        \intarray_gset:Nnn
          \g__pdfoutline_parent_id_intarray
          { \g__pdfoutline_id_int }
          { \g__pdfoutline_current_parent_id_tl }
        \intarray_gset:Nnn
          \g__pdfoutline_kid_count_intarray
          { \g__pdfoutline_current_parent_id_tl }
          { \intarray_item:Nn \g__pdfoutline_kid_count_intarray { \g__pdfoutline_current_pare
       }
       {
```

The first child of the previous bookmark

```
         \int_compare:nNnTF {#1} > {0}
          {
            \int_gincr:N \g__pdfoutline_current_level_int
            \tl_gset:Ne \g__pdfoutline_current_parent_id_tl
              { \int_eval:n { \g__pdfoutline_id_int -1 } }
            \intarray_gset:Nnn
              \g__pdfoutline_parent_id_intarray
               { \g__pdfoutline_id_int }
               { \g__pdfoutline_current_parent_id_tl }
            \intarray_gset:Nnn
              \g__pdfoutline_kid_count_intarray
              { \g__pdfoutline_current_parent_id_tl }
              { 1 }
           }
```

The most complicated case: going up the hierarchy. The root level is 1, so we can go up at most current level minus 1 steps. If we are already at the root level, the current parent is 1.

```
           {
            \int_compare:nNnTF { \g__pdfoutline_current_level_int } = {1}
             {
               \tl_gset:Ne \g__pdfoutline_current_parent_id_tl {1}
             }
             {
               \int_step_inline:nn
                 { \int_min:nn { \int_abs:n {#1} } { \g__pdfoutline_current_level_int -
1 } }
                 {
```

Get recursively the parent of the current parent and decrease the level.

```
                   \tl_gset:Ne \g__pdfoutline_current_parent_id_tl
                     { \intarray_item:Nn \g__pdfoutline_parent_id_intarray { \g__pdfoutline_
                   \int_gdecr:N \g__pdfoutline_current_level_int
                 }
             }
```

Store the parent and increase the kid count of the finally found parent (if this is the root level, the id is 1)

```
            \intarray_gset:Nnn
```

```
                \g__pdfoutline_parent_id_intarray
                 { \g__pdfoutline_id_int }
                 { \g__pdfoutline_current_parent_id_tl }
              \intarray_gset:Nnn
                \g__pdfoutline_kid_count_intarray
                { \g__pdfoutline_current_parent_id_tl }
                {
                  \intarray_item:Nn
                   \g__pdfoutline_kid_count_intarray
                   { \g__pdfoutline_current_parent_id_tl } + 1
                }
            }
        }
```

Now we store the collected data into the tl var.

```
    \__pdfoutline_store:nnn {#2}{#3}{#4}
  }
```

(*End of definition for* `\__pdfoutline_item:nnnn`.)

`\__pdfoutline_color_export:nnN` This exports space separated values from a color expression or a `[model]{values}` expression. It is used in the next command.

```
\cs_new_protected:Npn \__pdfoutline_color_export_aux:wnnN [#1] #2 #3 #4
  {
     \color_export:nnnN {#1}{#2}{ space-sep-#3 }#4
  }
\cs_new_protected:Npn \__pdfoutline_color_export:nnN #1 #2 #3 %#1 color, #2 target model, #3
  {
    \tl_if_blank:nTF { #1 }
     { \tl_clear:N #3 }
     {
       \tl_if_head_eq_charcode:nNTF {#1} [ %]
        {
          \__pdfoutline_color_export_aux:wnnN #1 { #2 } #3
        }
        {
          \color_export:nnN { #1 } { space-sep-#2 } #3
        }
     }
  }
\cs_generate_variant:Nn \__pdfoutline_color_export:nnN {ee}
```

(*End of definition for* `\__pdfoutline_color_export:nnN`.)

`\__pdfoutline_store:nnn` This command stores the data into the tl var. The backends have different requirements regarding the handling, so we keep arguments separate and generic to stay flexible.

```
\cs_new_protected:Npn \__pdfoutline_store:nnn #1 #2 #3
% #1: action or destination, #2: title, #3: keyword action or goto
  {
    \int_compare:nNnF { \g__pdfoutline_current_level_int } < { \l_pdfoutline_open_int }
      { \bool_set_false:N \l_pdfoutline_open_bool }
    \__pdfoutline_color_export:eeN {\l_pdfoutline_color_tl}{\l_pdfoutline_color_model_tl}\l__
```

```
    \tl_build_gput_right:Ne \g__pdfoutline_collect_build_tl
      {
        \exp_not:N \use:c { __pdf_backend_#3:nnnnnnn }
        % #1 level, #2 kid count, #3 open? #4 color, #5 flag, #6 action/destination, #7 title
          {
            \int_use:N \g__pdfoutline_current_level_int
          }
          {
            \exp_not:N \intarray_item:Nn
              \exp_not:N \g__pdfoutline_kid_count_intarray
              { \int_use:N \g__pdfoutline_id_int }
          }
          {
            \bool_if:NTF\l_pdfoutline_open_bool {\exp_not:N \c_true_bool }{ \exp_not:N \c_fal
          }
          {
            \l__pdfoutline_tmpa_tl
          }
          {
            \bitset_to_arabic:N \l_pdfoutline_F_bitset
          }
          {
            \exp_not:n {#1} % action/destination, or expand??
          }
          {
            \exp_not:n {#2} % title, or expand??
          }
       }
  }
```

(*End of definition for* `\__pdfoutline_store:nnn`.)

2.4 References

`\pdfoutline_id_ref_last:`

```
\cs_new:Npn \pdfoutline_id_ref_last: {\int_use:N\g__pdfoutline_id_int}
```

(*End of definition for* `\pdfoutline_id_ref_last:`. *This function is documented on page 4.*)

`\pdfoutline_level_ref_last:`

```
\cs_new:Npn \pdfoutline_level_ref_last: {\int_use:N\g__pdfoutline_current_level_int}
```

(*End of definition for* `\pdfoutline_level_ref_last:`. *This function is documented on page 4.*)

`\pdfoutline_parent_ref:n`

```
\cs_new:Npn \pdfoutline_parent_ref:n #1 { \intarray_item:Nn\g__pdfoutline_parent_id_intarray{
⟨/package⟩
```

(*End of definition for* `\pdfoutline_parent_ref:n`. *This function is documented on page 4.*)

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.

T

U