The l3pdffield-textfield module
Commands to create textfield form fields
LaTeX PDF management testphase bundle

The LaTeX Project*

Version 0.95c, released 2021-03-17

1 l3pdffield-textfield Introduction

Enter your name:

This is the documentation for textfield fields, for general information about form fields check the documentation l3pdffield.

Textfields allows the user to input text which is then rendered by the PDF viewer and often can also be saved together with the PDF.

Currently the package doesn't initialize the font `/Helv` used by default in the fields (It works also without it, but this isn't fully compliant.) I don't want to setup the same font twice and haven't yet decided how to organize the collaboration with hyperref. So for now you should load hyperref and issue the `\Form` command.

Please keep in mind

- Not every PDF viewer supports textfields.
- The font and the exact position of the chars depends on the PDF viewer.
- The handling can depend on settings in the PDF viewer. In adobe reader for example I had to disable an option to avoid that it tries to create an appearance itself
- Standards like pdf/A disable features of form fields too (as you typically can't change the PDF).

1.1 About fonts

The font color and font size can be changed with the keys `fontcolor` and `fontsize` described below. This works quite reliably (but e.g. the PDF viewer of my browser insists to show the font in blue while editing).

Much more difficult is the font family: a typical wish for textfields is that a font matching the document font is used in the fields. But how can that be done? Obviously it is not possible like with a checkbox to prepare and include appearances with the right look, the PDF reader has to render the text dynamically.

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

The PDF reference mentions only one way to pass a font setting to the text field: The `DA` key can contain a font operator and the fontname used should be added along with a object reference of the font to the form resources in the `AcroForm/DR/Font` dictionary. The question is: what does a PDF renderer make with this data? Clearly the main resources to render the text can not be to rely on an embedded font and a PDF font operator as both is not enough to guarantee a sensible text support: a font operator is normally used in the page stream along with glyph indices and positioning instructions, not with unicode input, and typically a font is only partially embedded. This means the PDF viewers and editors actually have to use external resources–system fonts, or fonts that come along with the viewer and libraries to handle font shaping.

Hyperref may add a reference the type1 font helvetica to the DA key, but PDF viewers use actually Arial. And if one doesn't add a font with `DA` and `DR/Font` the text field works nevertheless.

This doesn't mean that the DA/DR info is ignored altogether. Adobe Reader and Pro seems to use it to choose a fitting fallback font:

- One can change the font by simply using the font name (the examples use the `font` key of this module). One get courier with `font=courier`, times with `font=times`, comic sans with `font=ComicSansMS`, DejaVu with `font=DejaVuSans`. The fonts don't need to be used in the document or added as resources in the DR/Font dictionary. Some fonts work better than other: comic sans is quite unproblematic, but DejaVu seems to trigger a font search first and so is slower (perhaps the.
- If one fill and then save such a PDF at least adobe adds the missing font resources to the PDF.

This lazy font setting doesn't work with other PDF viewers. In other test I had better results by actually embedding a few glyphs of the font and then adding the font as resource to the DR/Font dictionary. Sadly this requires lualatex, only there is possible to retrieve the internal font name and font object number for a larger set of fonts—with pdflatex it works only for fonts not using a virtual font, and xelatex has no access at all. The code for lualatex looked like this (the font one wants to use should be active)

```
\pdffield_textfield:n {name=text,font=F\pdffeedback~fontname\font}
\pdfmanagement_add:nxx{Catalog/AcroForm/DR/Font}
      {F\pdffeedback~fontname\font}
      {\pdffeedback~fontobjnum\font \c_space_tl 0 \c_space_tl R}
```

1.2 Vertical alignment

There is no way to set a reference point for the baseline. Alignment must achieved by fiddling with the height and depth of the field. The defaults (which uses as height the fontsize, and a bit depth) works okay in adobe reader with arial. If the font is set to courier, the text jumps up and one has to decrease the height by around 30%. For a multiline field which should be top aligned you should increase the depth by the wanted amount *and* the height by around 2pt.

The geometry of the fields is seen by TeX, and so can influence the line spacing. If needed you should hide the height with `\raisebox` or `\smash`.

2 Textfields

Input a value:

2.1 Commands

\pdffield_textfield:n `\pdffield_textfield:n{⟨key val list⟩}`

This creates a textfield. The list of allowed keys is described below. The ⟨*key val list*⟩ should at least set the name, without it the default name `textfield` is used. Textfields with the same name belong to the same field and are filled together. The default appearance is a light gray background. The default appearance is setup at the first use.

2.2 Keys

The new textfield command accept all field and annot keys from l3pdffield, please check the documentation there. The list here mentions only a few central keys, and the keys which are either specific for textfields, or have changed functionality.

Disabled keys are

- `FT` is overwritten.
- `DA` is overwritten. Locally use `fontcolor`, `fontsize` and `font` instead. Globally you can set the `DA` key in the catalog with for example

  ```
  \pdfmanagement_add:nnn{Catalog/AcroForm}{DA}{(/Helv~10~Tf~1~1~0~rg)}
  ```

- For textfields only the field flags `ReadOnly`, `Required NoExport Multiline`, `Password`, `FileSelect`, `DoNotSpellCheck`, `DoNotScroll` and `Comb` make sense. If `Comb` is set, `Multiline`, `Password` and `FileSelect` are unset, `MaxLen` is set to 10 if it hasn't been set previously.

preset-textfield `preset-textfield = {⟨key-val-list⟩}`

This allows to set default keys for a textfield.

name `name = ⟨partial name⟩`
T `T = ⟨partial name⟩`

This sets like the `T` key for fields the partial name of the field. The value shouldn't contain a period, be not empty and sensibly consist of simple chars. Additionally the value is used to create the field ID. This means that textfield with the same partial name are annotations with the same field as parent and are filled together—this what is typically expected. The field ID is then internal and can not be used to attach another annotation. For explicit control of the field ID use the `fieldID` key.

fieldID `fieldID = ⟨field ID⟩`

For experts only! This allows to give the textfield a specific ID. This is only useful in the context of a larger fieldset or if you want to attach another annotation to the field with `\pdffield_annot:n`. If used wrongly you can easily create an invalid fieldset. It allows you to create to fields with the same partial name, but if you want to see both you need to ensure that their full names are different—for example by adding some parent fields.

parent `parent = ⟨field ID⟩`

This is only needed if the field should be part of a larger fieldset. The value should be a field ID of a field created previously with `\pdffield_field:nn`.

altname `altname = ⟨string⟩`
TU `TU = ⟨string⟩`

This is sets an alternative name for user interaction. This name can only be set at the first textfield instance, when the field is initialized.

mappingname `mappingname = ⟨string⟩`
TM `TM = ⟨string⟩`

This is sets an alternative name for export. This name can only be set at the first textfield instance, when the field is initialized.

width `width = ⟨dim expression⟩`
height `height = ⟨dim expression⟩`
depth `depth = ⟨dim expression⟩`

These keys allow to set the dimensions of textfield instance. The value should be a dimension expression. By default `width` is 3cm, the `height` uses the fontsize (`\f@size` in pt), the `depth` is 0.3`\f@size`.

appearance `appearance = ⟨name⟩`
rollover-appearance `rollover-appearance = ⟨name⟩`
down-appearance `down-appearance = ⟨name⟩`

This keys sets the normal appearance, the rollover appearance (when the mouse hovers over the textfield) and the down appearance (when the mouse clicks). They expect the name of an existing form Xobject as value. The initial value is `pdffield/textfield/default` for the normal appearance and shows a light gray background. The other appearance are not set by default (and it is quite unclear if they make sense for a textfield).

fontcolor `fontcolor = ⟨color expression⟩ | [⟨model⟩]{⟨values⟩}`

This is sets the color of font in the textfield. Internally currently RGB is used. The colors used in ⟨*color expression*⟩ must be known to the l3color commands. The initial color is black. It is a *field* setting, that means instances share the color.

fontsize `fontsize = ⟨dim expression⟩`

This is sets the size of the font in the textfield. The default value is the current font size (`\f@size` in pt). It is a *field* setting, that means instances share the size.

font `font = ⟨symbolic font name⟩`

This is sets the font in the textfield. See the discussion at the begin of the documentation about some background. The default value is `Helv` and this name is setup if you load hyperref and issue the `\Form` command. The default value is the current font size (`\f@size` in pt). It is a *field* setting, that means instances share the font.

2.3 Using with hyperref

The `\TextField` command from hyperref also prints a label, something that the command here doesn't do. A redefinition like the following should allow `\TextField` to use the commands of this module. Be aware that the behaviour will not be identical! Not every setting and key from hyperref has been copied.

```
\ExplSyntaxOn\makeatletter
\def\@TextField[#1]#2{\LayoutTextField{#2}{\pdffield_textfield:n {name=#2,#1}}}
\ExplSyntaxOff\makeatother
```

3 l3pdffield-textfield Implementation

```
⟨*package⟩
⟨@@=pdffield⟩
```

3.1 Variables

```
\tl_new:N  \l__pdffield_DA_fontcolor_tl
\dim_new:N \l__pdffield_DA_fontsize_dim
\tl_new:N  \l__pdffield_DA_fontname_tl
```

3.2 Messages

3.3 Appearances

The default appearances are a cross (`\texttimes`), Every appearance should have two versions and follow the naming module/⟨*name*⟩/Yes and module/⟨*name*⟩/Off.

__pdffield/textfield/default_appearances:

This defines the standard appearance. It is setup at the first use of a textfield.

```
\cs_new_protected:cn {__pdffield/textfield/default_appearance:}
  {
    \pdffield_appearance:nn {pdffield/textfield/default}
     {
       { \color_select:n{black!5!white}\rule{\paperwidth}{\paperheight} }
     }
    \cs_gset_eq:cN {__pdffield/textfield/default_appearance:} \prg_do_nothing:
  }
```

(*End definition for* `__pdffield/textfield/default_appearances:`.)

3.4 Creating the field

A field should be created if the name doesn't exist

```
\cs_new_protected:Npn \__pdffield_textfield_field:n #1 %name
  {
    \pdf_object_if_exist:nF {__pdffield/field/__pdffield/textfield/#1}
      {
        \__pdffield_field:n { __pdffield/textfield/#1 }
      }
    \keys_set:nn {pdffield}{parent=__pdffield/textfield/#1}
  }
\cs_generate_variant:Nn \__pdffield_textfield_field:n {V}
```

3.5 Assembling the textfield

__pdffield_textfield:n

```
\cs_new_protected:Npn \__pdffield_textfield:n #1
  {
    \group_begin:
    \cs_set_eq:NN\__pdffield_value_handler:nN \__pdffield_textfield_value_handler:nN
    \use:c {__pdffield/textfield/default_appearance:}
```

Setting up the defaults.

```
    \keys_set:nn {pdffield}
      {
        ,fieldID=
        ,name=textfield
        ,appearance = pdffield/textfield/default
        ,width  = 3cm
        ,fontsize= \f@size pt
        ,height =  \f@size pt,
         depth  =   \fp_eval:n {0.3*\f@size} pt,
        % font defaults!!
      }
    \keys_set:nn { pdffield }{__pdffield/preset/textfield,#1}
    \int_compare:nNnT {\bitset_item:Nn \l__pdffield_Ff_bitset {Comb}}={1}
      {
        % warning if set?
        \keys_set:nn { pdffield }
          {
            ,unsetFf={FileSelect,Multiline,Password}
          }
        \pdfdict_get:nnN {l__pdffield/field}{MaxLen}\l__pdffield_tmpa_tl
        \quark_if_no_value:NT \l__pdffield_tmpa_tl
         {
           \keys_set:nn { pdffield}
            {
              MaxLen = 10 %variable
            }
            % warning
         }
      }
    \keys_set:nn { pdffield }
      {
        ,FT= Tx
        ,AS=
        ,DA=
           {
              \pdf_name_from_unicode_e:n{\l__pdffield_DA_fontname_tl}
              \c_space_tl
              \dim_to_decimal_in_bp:n{\l__pdffield_DA_fontsize_dim}
              \c_space_tl
              Tf
              \c_space_tl
              \l__pdffield_DA_fontcolor_tl
              \c_space_tl
              %\l__pdffield_text_DAextra_tl
```

```
            }
        }
      \tl_if_empty:NT\l__pdffield_fieldID_tl
        {
          \pdfdict_get:nnN {l__pdffield/field}{T}\l__pdffield_fieldID_tl
          \tl_put_left:Nn \l__pdffield_fieldID_tl {__pdffield/textfield/}
        }

      \__pdffield_textfield_field:V\l__pdffield_fieldID_tl
      \__pdffield_annot:
      \group_end:
    }
```

(*End definition for* `\__pdffield_textfield:n`.)

3.6 Keys

Most keys are inherited simply the ones from the generic field and annot keys. A key to decide if the box is initially checked or not. We stay in the same family so that we can build a style.

```
\keys_define:nn {pdffield}
  {
    ,fontcolor .code:n =
       {
         \__pdffield_color_set:nn {__pdffield/tmp}{#1}
         \color_export:nnN{__pdffield/tmp}{space-sep-rgb}\l__pdffield_DA_fontcolor_tl
         \tl_put_right:Nn \l__pdffield_DA_fontcolor_tl{~rg}
       }
    ,fontcolor .initial:n = black,
    ,fontcolor .groups:n = {textfield}
    ,font .tl_set:N      = \l__pdffield_DA_fontname_tl
    ,font .initial:n     = {Helv}
    ,font .groups:n = {textfield}
    ,fontsize .dim_set:N = \l__pdffield_DA_fontsize_dim
    ,fontsize .initial:n = {10bp}
    ,fontsize .groups:n = {textfield}
  }
```

And a key to set a dedicated field ID

```
\keys_define:nn { pdffield }
 {
   fieldID .tl_set:N = \l__pdffield_fieldID_tl
 }
```

3.7 Handler

```
\cs_new_protected:Npn \__pdffield_textfield_value_handler:nN #1#2
  {
     \pdf_string_from_unicode:nnN {utf8/string}{#1}#2
  }
```

3.8 user commands

\pdffield_textfield:n

```
\cs_set_eq:NN \pdffield_textfield:n \__pdffield_textfield:n
⟨/package⟩
```

(*End definition for* `\pdffield_textfield:n`. *This function is documented on page 3.*)