AcroTEX.Net

The grayhints Package

D. P. Story

Prepared: March 20, 2017

www.acrotex.net
Version v1.0, 2017/03/02

The grayhints Package

Table of Contents

1. Introduction

We often see in HTML pages or in compiled executable applications, form fields (text fields, input fields) that require user input. The untouched field has text within it informing the user of the nature of the data to be entered into the field. This, usually, grayed hint immediately disappears when the user focus the cursor on the field. We illustrate the concept with an example or two.

First Name

yyyy/mm/dd Reset

Of course, the usual tooltips may also be provided.

It is not natural for Adobe form fields to do this, it takes some support code for it to work properly; scripts for the Keystroke, Format, OnFocus, and OnBlur events are needed.

2. Package options

Without passing any options, the eforms package of AeB, dated 2017/02/27, is required and a document JavaScript function `AllowCalc()` is automatically embedded in the document; however there are options to modify this default setup.

`usehyforms` By default, this package requires eforms, dated 2017/02/27; however, if you are more comfortable using the form fields of hyperref, specify the option `usehyforms`.[1] When `usehyforms` is specified, insdljs dated 2017/03/02 or later is required. This requirement is to support the `usealtadobe`, discussed next.

`nocalcs` If this option is taken, the document JavaScript function `AllowCalc()` is not embedded in the document. The implications are that you are not using any calculation fields.

`usealtadobe` If you have the Acrobat application, you can edit form fields. When you write custom formatting scripts (as does this package) using Adobe's built-in functions, such as `AFNumber_Keystroke` and `AFNumber_Format`, the user-interface for editing the custom script is not available. The `usealtadobe` option is passed to insldjs; insdljs, in turn, inputs alternate names for the common Adobe built-ins. Refer to Section 3.4 for more information.

`nodljs` When this option is specified, there are no requirements placed on this package; that is, neither eforms nor insdljs are required.

Demo file: `gh-eforms.tex, gh-hyperref.tex`. The latter file uses the `usehyforms` option (and hyperref form fields), while the former uses the eforms package.

[1] eforms and hyperref form fields can be used in one document.

3. Creating a form field with a gray hint

In this documentation, we use eforms form fields to illustrate concepts, the demonstration file gh-hyperref.tex has the form field markup for the case of hyperref forms.

There are two cases: (1) an ordinary variable text form field (this includes text fields and editable combo boxes) with no calculate script; (2) same as (1), but the field has a calculate script.

3.1. Variable text field, no calculate script

When there is no calculate script, to obtain a gray hint, it is necessary to supply scripts for the Format, Keystroke, OnFocus, and OnBlur events. The scripts are all defined in the grayhints package. In addition, the color of the text in the text field must be appropriate. We illustrate,

```
1 \textField[\TU{Enter your first name so I can get to know you better}
2     \textColor{\matchGray}\AA{%
3     \AAKeystroke{\KeyToGray}
4     \AAFormat{\FmtToGray{First Name}}
5     \AAOnFocus{\JS{\FocusToBlack}}
6     \AAOnBlur{\JS{\BlurToBlack}}
7 }]{NameFirst}{2in}{11bp}
```

By default, the text color is black and the grayed hint text is light gray. The tool tip (\TU) is grayed out, as it is optional. In line (2) we match the color for the text to the gray color using the command \matchGray of grayhints. Within the argument of \AA, the \AAFormat, \AAKeystroke, \AAOnFocus, and \AAOnBlur scripts are inserted.

Keystroke Script: In line (3), \KeyToGray is placed within the argument of \AAKeystroke. This script changes the color of the text to gray when the field is empty.

Format Script: The script snippet \FmtToGray takes a single argument, which is the text of the hint. In line (4) the hint is 'First Name'.

OnFocus Script: The code snippet \FocusToBlack is inserted into the argument of \OnFocus, as seen in line (5). When the field comes into focus, this script changes the color to the normal color (usually black).

OnBlur Script: In line (6), the \BlurToBlack script is placed within the argument of \OnBlur, in the manner indicated. When the field loses focus (is blurred), the script changes the color of text to gray if the field is empty or to its normal color (usually black), otherwise.

The hyperref form field counterpart to the above example is,

```
1 \TextField[name={NameFirst},
2     height=11bp,width=2in,
3     color=\matchGray,
4     keystroke=\KeyToGray,
5     format=\FmtToGray{First Name},
6     onfocus=\FocusToBlack,
7     onblur=\BlurToBlack]{}
```

The two fields appear side-by-side:

First Name First Name Reset

Both fields appear in the ‘default’ appearance.

3.2. Variable text field, with calculate script

If you want to make calculations based on entries in other fields, you will need the code snippet `\CalcToGray` as part of your calculate script.

```
1 \textField[\TU{The total for first and second integers}
2     \textColor{\matchGray}\AA{%
3     \AAKeystroke{AFNumber_Keystroke(0,1,0,0,"",true);\r\KeyToGray}
4     \AAFormat{AFNumber_Format(0,1,0,0,"",true);\r\FmtToGray{Total}}
5     \AACalculate{var cArray=new Array("Integer");\r
6         if(AllowCalc(cArray))AFSimple_Calculate("SUM", cArray );\r
7         \CalcToGray}
8     \AAOnFocus{\JS{\FocusToBlack}}
9     \AAOnBlur{\JS{\BlurToBlack}}}
10 ]{TotalNumbers}{1in}{11bp}
```

The use of `\r` is optional, the author uses this to format the script within the user-interface of Acrobat. The `\textColor` (line (2)), `\AAOnFocus` (line (8)), and `\AAOnBlur` (line (8)) are the same as earlier presented. Several comments are needed for the `\AAKeystroke`, `\AAFormat` and `\AACalculate` lines.

- This is a number field, so we use the built-in functions `AFNumber_Keystroke` and `AFNumber_Format` provided by the Adobe Acrobat and Adobe Acrobat Reader distributions. In lines (3) and (4), the `\KeyToGray` and `\FmtToGray` code snippets follow the built-ins.[2]
- For the Calculate event, special techniques are used. We define an array `cArray` (line (5)) consisting of the names of all the dependent fields we use to calculate the value of this field. In line (6), we make the calculation (`AFSimple_Calculate`) only if the document JavaScript function `AllowCalc(cArray)` returns true. The function returns true only if at least one of the fields is not empty. Following the calculation comes the code snippet `\CalcToGray`; this changes the text color to gray if the field is empty and to the normal color (usually black) otherwise.

  The function `AllowCalc()` is defined for all options except for the `nodljs` option.

Let’s go to the examples. Build three fields (four actually), in the first two enter integers, the other two fields compute their sum.

① First Integer
② Second Integer
③ Total
④ Total Reset

[2] As a general rule, the code snippets `\KeyToGray`, `\FmtToGray`, and `\CalcToGray` should inserted after any built-in functions.

Enter numbers into the first two text fields (① and ②), the totals of these two fields appear in the last two fields (③ and ④). Total field ③ uses the recommended script `if(AllowCalc(cArray)` (see line (6) above), whereas field ④ does not. Initially, they both behave the same way until you press the reset button. For field ③ the gray hint appears, for field ④ the number zero (0) appears. This is because the calculation was allowed to go forward, and the calculated value is zero even through none of the dependent fields have a value. If you want the gray hint in the total field, you must use the conditional `if(AllowCalc(cArray)`.[3]

3.3. Changing the colors for gray hints

For the fields in which the gray hint scripts are used, there are two colors that are relevant, the normal color (defaults to black) and the gray color (defaults to light gray). The command `\normalGrayColors{⟨normalcolor⟩}{⟨graycolor⟩}` sets this pair of colors. The arguments for `\normalGrayColors` are JavaScript colors; they may be in any of the following four forms: (1) a JavaScript color array `["RGB",1,0,0]`; (2) a predefined JavaScript color, such as `color.red`; (3) a declared (or named) LaTeX color such as `red`; or (4) a non-declared LaTeX color such as `[rgb]{1,0,0}`. If the package `xcolor` is not loaded, only methods (1) and (2) are supported.

The package default is `\normalGrayColors{color.black}{color.ltGray}`. The predefined JavaScript colors are,

Color Models		
GRAY	RGB	CMYK
`color.black`	`color.red`	`color.cyan`
`color.white`	`color.green`	`color.magenta`
`color.dkGray`	`color.blue`	
`color.gray`		
`color.ltGray`		

All these colors are defined in the LaTeX color packages, except for possibly `dkGray`, `gray`, and `ltGray`. These three are defined in `grayhints`.

We repeat the 'First Name' example with different declared colors. We begin by declaring,

```
\normalGrayColors{blue}{magenta}
```

then build a 'gray hinted' field,

First Name Reset

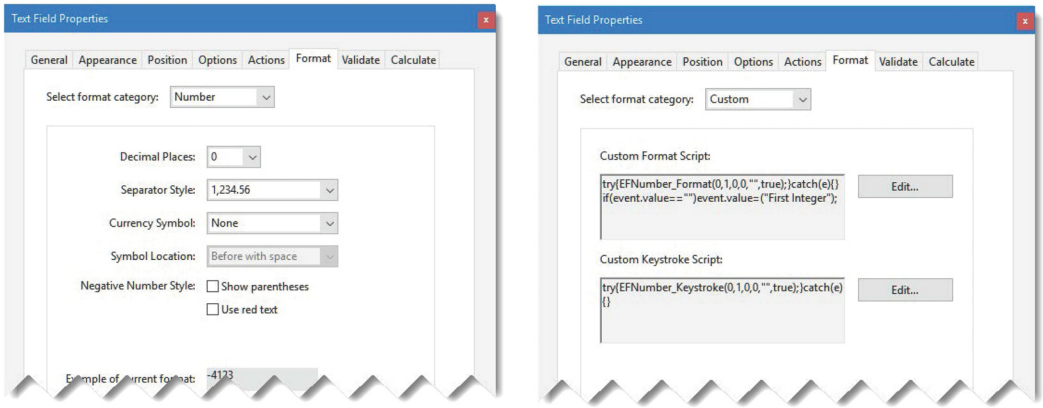
3.4. Remarks on the `usealtadobe` option

The `usealtadobe` option is useful for developers who have the Adobe application and who wish to develop and test scripts that extend in the current work. The `usealtadobe` option inputs from `insdljs` the following alternate names. As a general rule, all Adobe

[3] Hence, don't use the `nodljs` option.

built-in format, validate, and calculation functions that begin with 'AF' are given alternate names that begin with 'EF'. More specifically, the table below lists the effected functions.

Adobe function name	**Alternate function name**
AFNumber_Keystroke	EFNumber_Keystroke
AFNumber_Format	EFNumber_Format
AFPercent_Keystroke	EFPercent_Keystroke
AFPercent_Format	EFPercent_Format
AFDate_Format	EFDate_Format
AFDate_Keystroke	EFDate_Keystroke
AFDate_FormatEx	EFDate_FormatEx
AFTime_Keystroke	EFTime_Keystroke
AFTime_Format	EFTime_Format
AFTime_FormatEx	EFTime_FormatEx
AFDate_KeystrokeEx	EFDate_KeystrokeEx
AFSpecial_Keystroke	EFSpecial_Keystroke
AFSpecial_Format	EFSpecial_Format
AFSpecial_KeystrokeEx	EFSpecial_KeystrokeEx
AFRange_Validate	EFRange_Validate
AFRange_Validate	EFRange_Validate
AFSimple_Calculate	EFSimple_Calculate
AFMergeChange	EFMergeChange



(a) Using AFNumber_Format (b) Using EFNumber_Format

Figure 1: Format tab: 'AF' versus 'EF' functions

Figure 1 shows the impact of using the 'EF' functions. On the left, AFNumber_Format is used to format a number field that uses gray hints using the code

```
AFNumber_Format(0,1,0,0,"",true)\r\FmtToGray
```

As can be seen in sub-figure (a), or more accurately not seen, the code is not seen through the user-interface of Acrobat. In sub-figure (b) the underlying code is seen (and therefore editable through the user-interface) because the 'EF' version of the function was used:

```
\try{EFNumber_Format(0,1,0,0,"",true)}catch(e){}\r\FmtToGray
```

Note this code is wrapped in a `try/catch` construct; this is optional. The insdljs package defines a helper command `\dlTC` to do the wrapping for you:

```
\dlTC{EFNumber_Format(0,1,0,0,"",true)}\r\FmtToGray
```

When using pdflatex or xelatex, `try/catch` appears not to be needed, but when Adobe Distiller is used, Acrobat throws an exception when the file is first created. The `try/catch` suppresses (catches) the exception.

4. My retirement

Now, I simply must get back to it. ☺