grayhints: Create gray hints in text fields

D. P. Story
Email: dpstory@acrotex.net

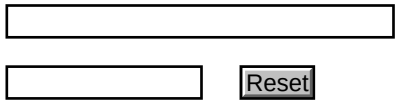
processed November 2, 2018

Contents


```
1 %<*package>
```

1 Description

We often see in HTML pages or compiled executable applications, form fields (text fields, input fields) that require user input. The untouched field has text within it that informs the user of the nature of the data to be entered into the field. This 'grayed hint' immediately disappears when the user focuses the cursor on the field. Lest I be accused of being too obtuse, we illustrate with an example or two.



Of course, the usual tool tips may also be provided.

It is not natural for Adobe form fields to do this, it takes a lot of support code for it to work properly; the Keystroke, Format, OnFocus, OnBlur, and Calculate events are needed. The verbatim listing of the first example field above is,

```
\textField[\textColor{\matchGray}
   \TU{Enter your first name so I can get to know you better}
   \AA{\AAFormat{\FmtToGray{First Name}}
   \AAKeystroke{\KeyToGray}
```

```
    \AAOnFocus{\JS{\FocusToBlack}}
    \AAOnBlur{\JS{\BlurToBlack}}
    \AACalculate{\CalcToGray} %<- required if using PDF-XChange Editor
}]{NameFirst}{2in}{11bp}
```

Code snippets are inserted into the Keystroke, Format, OnFocus, OnBlur, and Calculate events.

Demo files: Four sample files are provided: `gh-eforms.tex`, `gh-hyperref.tex`, `gh-fmts-eforms.tex`, and `gh-fmts-hyperref.tex`.

2 Documentation and Code

The eforms package is preferred, but you can use the form field macros of hyperref. Any unrecognized options specified for the grayhints package are passed on to insdljs. If the document author does not want to use eforms, he/she can pass the

`usehyforms` option `usehyforms` to use the form fields of hyperref, in this case insdljs is required. For the last option, `nodljs` is for users of hyperref forms who do not want to use insdljs. In the latter case, the option `usehyforms` should not be used for that will include insdljs.

```
\DeclareOption{usehyforms}{%
    \def\FormsRequirement{\RequirePackage{insdljs}[2017/03/02]}}
\def\FormsRequirement{\RequirePackage{eforms}[2017/02/27]}
```

`usealtadobe` The `usealtadobe` option is deprecated, the function definitions are automatically loaded unless the `nocalcs` or `nodljs` option is taken.

```
\DeclareOption{usealtadobe}{\PackageWarningNoLine{grayhints}
  {The 'usealtadobe' option is now deprecated.\MessageBreak
   The alternate functions are automatically loaded.\MessageBreak
   Please remove this grayhints package option}}
```

`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.

```
\DeclareOption{nocalcs}{\let\nocalcs\endinput}
\let\nocalcs\relax
```

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

```
\DeclareOption{nodljs}{\let\FormsRequirement\relax
  \let\nodljsend\endinput}
\let\nodljsend\relax
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{insdljs}}
\ProcessOptions
\FormsRequirement
```

We include `eqcolor.def`, a component of exerquiz to help parse colors, and to match JS colors with PDF colors.

```
\@ifundefined{jsColor}{\let\eq@YES=y\let\eq@NO=n%
```

```
  \InputIfFileExists{eqcolor.def}
  {\PackageInfo{grayhints}{Inputting eqcolor.def from exerquiz}}
  {\PackageError{grayhints}{cannot find eqcolor.def belonging
  to exerquiz}{Refresh your file name database and try again.}}
}{}
```

2.1 JavaScript snippets for Field JavaScript

Code snippets are inserted in to the Format, Calculate, OnFocus, and OnBlur events, as illustrated above.

`\normalGrayColors{`⟨*normalcolor*⟩`}{`⟨*graycolor*⟩`}` There are two colors in play, the normal color of the text field (⟨*normalcolor*⟩) and the color of the "grayed" text (⟨*graycolor*⟩). We set the defaults to `color.black` and `color.ltGray`, respectively. The two parameters are JavaScipt colors: array type `["RGB", 1, 0, 0]`, or predefined type `color.blue`. The command `\jsColor` is used to assign colors (taken from `eqcolor.def` of exerquiz).

```
\newcommand{\normalGrayColors}[2]{\def\gh@rgi{#1}\def\gh@rgii{#2}%
  \ifx\gh@rgi\@empty\else
    \jsColor\gh@normalcolor{#1}\gh@chkTr@nsparency\fi
  \ifx\gh@rgii\@empty\else\jsColor\gh@graycolor{#2}\m@tchGray\fi}
\def\gh@normalcolor{}\def\gh@graycolor{}
\AtEndOfPackage{\normalGrayColors{color.black}{color.ltGray}}
```

There are several predefined JavaScript colors the user can specify. We need to convert them to PDF color too.

```
\definecolor{ltGray}{gray}{0.75}
\definecolor{gray}{gray}{.5}
\definecolor{dkGray}{gray}{.25}
\def\gh@pd@transparent{ltGray}\def\gh@transparent{transparent}
\def\gh@pd@black{black}\def\gh@pd@white{white}
\def\gh@pd@dkGray{dkGray}\def\gh@pd@gray{gray}\def\gh@pd@ltGray{ltGray}
\def\gh@pd@red{red}\def\gh@pd@green{green}\def\gh@pd@blue{blue}
\def\gh@pd@cyan{cyan}\def\gh@pd@magenta{magenta}
\def\gh@pd@yellow{yellow}
```

Convert the JS color for `\gh@graycolor` to a matching PDF color that can be used by the `\textColor` property of a form field. Here, we use commands defined in the `eqcolor.def` file from exerquiz. This command defines the user command `\matchGray`.

`\matchGray`

```
\def\m@tchGray{\eq@checkRawJSColor{\gh@graycolor}%
  \ifx\eqpredefineJSCol\eq@NO
```

`\gh@graycolor` is a JavaScript array

```
    \let\matchGray\@empty
    \expandafter\gh@extr@ctJSModelInfo\gh@graycolor\@nil
    \ifx\@rgi\@empty\else\edef\matchGray{\@rgi}\fi
    \ifx\@rgii\@empty\else\edef\matchGray{\matchGray\space\@rgii}\fi
    \ifx\@rgiii\@empty\else
```

```
      \edef\matchGray{\matchGray\space\@rgiii}\fi
    \ifx\@rgiv\@empty\else\edef\matchGray{\matchGray\space\@rgiv}\fi
  \else
```

\gh@graycolor is a predefined color (color.ltGray)

```
    \expandafter\gh@getColorFromPrefined\gh@graycolor\@nil
    \@ifundefined{gh@pd@\pd@color}{%
      \def\gh@graycolor{color.ltGray}\def\pd@color{ltGray}%
      \PackageWarning{grayhints}
        {The color.\pd@color\space is undefined,\MessageBreak
        substituting color.ltGray}}{\ifx\pd@color\gh@transparent
      \def\gh@graycolor{color.ltGray}\def\pd@color{ltGray}%
      \PackageWarning{grayhints}
        {A transparent color is not supported,\MessageBreak
        using color.ltGray instead}\fi
    }%
    \edef\matchGray{\@nameuse{gh@pd@\pd@color}}%
  \fi}
\def\gh@chkTr@nsparency{\eq@checkRawJSColor{\gh@normalcolor}%
    \ifx\eqpredefineJSCol\eq@YES
        \expandafter\gh@getColorFromPrefined\gh@normalcolor\@nil
        \@ifundefined{gh@pd@\pd@color}{\def\gh@normalcolor{color.black}%
            \PackageWarning{grayhints}
            {The color.\pd@color\space is undefined,\MessageBreak
            substituting color.black}}{}%
        \ifx\pd@color\gh@transparent\def\gh@normalcolor{color.black}%
        \PackageWarning{grayhints}
            {A transparent color is not supported,\MessageBreak
            using color.black instead}\fi
    \fi
}
```

Various supporting macros to extract information.

```
\def\gh@extr@ctJSModelInfo[#1,#2]\@nil{%
  \gh@getspecv@lues#2,,,,\@nil}%
\def\gh@getspecv@lues#1,#2,#3,#4,#5\@nil{%
  \def\@rgi{#1}\def\@rgii{#2}\def\@rgiii{#3}\def\@rgiv{#4}}
\def\gh@getColorFromPrefined color.#1\@nil{\def\pd@color{#1}}
\def\gh@PriorFormat{event.target.savevalue=event.value;\jsR}
```

\FailStringDef provides a helpful string when the user enters an improper string. This is a language localization command.

```
\newcommand\FailStringDef{continue editing}
```

\EnterCommitFailDef Sets the code for \FailStringDef.

```
\newcommand\EnterCommitFailDef{event.value=("\FailStringDef");}
```

\EnterCommitFailEvent Sets the action when the user uses the Enter key to commit date, and the data is not validated.

```
\def\EnterCommitFailEvent#1{\def\@rgi{#1}\ifx\@rgi\@empty
  \def\gh@ECFE{\EnterCommitFailDef}\else\def\gh@ECFE{#1}\fi}
\EnterCommitFailEvent{}
```

`\CommitSuccessEvent` Set what happens when entering a valid string into a formatting text field

```
\def\CommitSuccessEvent#1{\def\@rgi{#1}\ifx\@rgi\@empty
\let\gh@CSE\jsR\else\def\gh@CSE{\jsR\jsT #1\jsR}\fi}
\let\gh@CSE\jsR
```

`\FmtToGray{`⟨*grayhint*⟩`}` This command is placed in the Format event. It places the hint ⟨*grayhint*⟩ as the formatting text string when the field is empty. If a built-in Adobe function is also used, use `\FmtToGray` after it; for example,

```
\AAFormat{AFNumber_Format(0,1,0,0,"",true);
\FmtToGray{⟨grayhint⟩}}
```

```
\newcommand\FmtToGray[1]{%
  if(typeof event.target.savevalue=="undefined"){\jsR\jsT
   event.target.savevalue="";\jsR\jsT
   event.target.success=false;\jsR
  }\jsR
  if(typeof event.target.ghBuiltin=="undefined")%
    event.target.ghBuiltin=false;\jsR
  if(!event.target.ghBuiltin)\gh@PriorFormat
  event.target.ghBuildtin=false;\jsR\gh@FmtToGray{#1}}
\def\gh@FmtToGray#1{%
  if(typeof event.target.saverc=="undefined")%
    event.target.saverc=true;\jsR
  if(!event.target.saverc||event.value==""){\jsR\jsT
    if(event.target.savevalue!=""&&%
      !event.target.success){\jsR\jsT\gh@ECFE\jsR\jsT
```

Here is the only PDF dependent code. The event sequence of close to but not exactly the same as the event sequence for the Adobe PDF viewers, we must insert the following two lines of code to make things work for PDF-XChange Editor.

```
      if(typeof app.pxceInfo!="undefined")%
        event.target.savevalue="";\jsR\jsT
    } else {\jsR\jsT\jsT
      event.target.success=false;\jsR\jsT\jsT
      event.value=("#1");\jsT\gh@CSE\jsT}\jsR
  } else {\jsR\jsT
      event.target.success=true;\gh@CSE}
}
```

`\KeyToGray` This command is placed in the Keystroke event. It changes the color to 'gray' (`\gh@graycolor`) if the field is empty. If a built-in Adobe function is also used, use `\KeyToGray` after it; for example,

```
\AAFormat{AFNumber_Keystroke(0,1,0,0,"",true);
\KeyToGray}
```

```
\newcommand\KeyToGray{%
  if(event.willCommit&&event.value!=""&&!event.rc)\jsR\jsT
   event.target.success=false;\jsR
  event.target.saverc=event.rc;\jsR
```

```
  event.rc=true;\jsR
  if(event.willCommit&&event.value=="")%
    event.target.textColor=\gh@graycolor;\jsR
  if(event.willCommit)%
    event.target.savevalue=event.value;
}
```

\CalcToGray The \CalcToGray is a Calculate script, it is needed only in a form field that performs a calculation. If a built-in Adobe function is also used, use \KeyToGray after it; for example,

```
\AACalculate{var cArray=new Array("Integer");\jsR
if (AllowCalc(cArray)) AFSimple_Calculate("SUM", cArray );\jsR
\CalcToGray}
```

If the target population might use PDF-XChange Editor, whose features closely mimic those of Adobe Acrobat Reader, the use of \CalcToGray is recommended in all fields.

```
\newcommand\CalcToGray{event.target.textColor=%
  (event.value=="")?\gh@graycolor:\gh@normalcolor;}
```

\FocusToBlack A command placed within the OnFocus event. When the field comes into focus, and the field is empty, the color for the text is turned to black. This can be redefined to another color.

(2018/10/04) We increase the complexity with the goal of getting a better user experience.

```
\newcommand\FocusToBlack{%
  if (typeof event.target.success=="undefined")%
    event.target.success=false;\jsR
  if(event.target.valueAsString==""%
    ||!event.target.success)\jsR\jsT
    event.target.textColor=\gh@normalcolor;
}
```

\BlurToBlack A command placed within the OnBlur event. It sets the text color to gray or black, depending on whether the field is empty or not. My be redefined with different colors.

(2018/10/04) We increase the complexity with the goal of getting a better user experience.

```
\newcommand\BlurToBlack{%
  if (!event.target.success||event.target.valueAsString=="")\jsR\jsT
   this.resetForm(event.target.name);\jsR
  event.target.savevalue="";\jsR
  event.target.textColor=(!event.target.success||%
  event.target.valueAsString=="")?\gh@graycolor:\gh@normalcolor;
}
```

2.2 LaTeX commands for built-in functions

We define a series of commands as a convenience to the user. The arguments of each are the JavaScript argument enclosed in parentheses.

`\NumKey` `\NumFmt` `\NumKey` for processing keystrokes for numbers, and `\NumFmt` formats a number according to its arguments.

```
\def\NumKey{EFNumber_Keystroke}
\def\NumFmt(#1){try{EFNumber_Format(#1)}catch(e){}}
```

`\DateKey` `\DateFmt` `\DateKey` and `\DateFmt` process the keystroke and format events for a date.

```
\def\DateKey{EFDate_Keystroke}
\def\DateFmt(#1){%
  AFDate_Format(#1);\jsR
  event.target.ghBuiltin=true;
}
```

`\DateKeyEx` `\DateFmtEx` `\DateKeyEx` and `\DateFmtEx` process the keystroke and format events for a date.

```
\def\DateKeyEx{EFDate_KeystrokeEx}
\def\DateFmtEx(#1){%
  AFDate_FormatEx(#1);\jsR
  event.target.ghBuiltin=true;
}
```

`\PercentKey` `\PercentFmt` `\PercentKey` and `\PercentFmt` process the keystroke and format events for a number represented as a percentage.

```
\def\PercentKey{EFPercent_Keystroke}
\def\PercentFmt(#1){%
```

Avoid the dreaded "0.00%" when the field is blank

```
  if(event.value!=""||%
    (typeof event.target.savevalue!="undefined"&&%
      event.target.savevalue!=""))%
    AFPercent_Format(#1);\jsR
  event.target.ghBuiltin=true;
}
```

`\TimeKey` `\TimeFmt` `\TimeFmtEx` `\TimeKey`, `\TimeFmt`, and `\TimeFmtEx` process the keystroke and format events for a time.

```
\def\TimeKey{EFTime_Keystroke}\def\TimeFmt{EFTime_Format}
\def\TimeFmtEx(#1){try{EFTime_FormatEx(#1)}catch(e){}}
```

`\SpecialKey` `\SpecialKeyEx` `\SpecialFmt` `\SpecialKey`, `\SpecialKeyEx`, and `\SpecialFmt` process the keystroke and format events for a special format.

```
\def\SpecialKey{EFSpecial_Keystroke}
\def\SpecialKeyEx{EFSpecial_KeystrokeEx}
\def\SpecialFmt(#1){try{EFSpecial_Format(#1)}catch(e){}}
```

`\RangeValidate` `\SimpleCalc` `\MergeChange` `\RangeValidate`, `\SimpleCalc`, and `\EFMergeChange` are specialized JS functions for setting a range resstriction in the validate event, for making a simple calculation in the calculate event, and a general purpose function to merging the current keystroke with event.value, valid for the keystroke event.

```
\def\RangeValidate{EFRange_Validate}
\def\SimpleCalc{EFSimple_Calculate}
\def\MergeChange{EFMergeChange}
```

2.3 Document JavaScript to support gray hints

The alternate names adobe built-in need to be used for any format function; the normal built-in function names can be otherwise be used.

```
\nodljsend
\begin{insDLJS}{altadbfncs}{gh: Support for Adobe built-in functions}
var EFNumber_Keystroke=AFNumber_Keystroke;
function EFNumber_Format(){
  event.target.savevalue=event.value;
  event.target.ghBuiltin=true;
  AFNumber_Format.apply(null,arguments);
}
var EFDate_Keystroke=AFDate_Keystroke;
function EFDate_Format(){
  event.target.savevalue=event.value;
  event.target.ghBuiltin=true;
  AFDate_Format.apply(null,arguments);
}
var EFDate_KeystrokeEx=AFDate_KeystrokeEx;
function EFDate_FormatEx(){
  event.target.savevalue=event.value;
  event.target.ghBuiltin=true;
  AFDate_FormatEx.apply(null,arguments);
}
var EFPercent_Keystroke=AFPercent_Keystroke;
function EFPercent_Format(){
  event.target.savevalue=event.value;
  event.target.ghBuiltin=true;
  AFPercent_Format.apply(null,arguments);
}
var EFTime_Keystroke=AFTime_Keystroke;
function EFTime_Format(){
  event.target.savevalue=event.value;
  event.target.ghBuiltin=true;
  AFTime_Format.apply(null,arguments);
}
function EFTime_FormatEx(){
  event.target.savevalue=event.value;
  ghBuiltin=true;
  AFTime_FormatEx.apply(null,arguments);
}
var EFSpecial_Keystroke=AFSpecial_Keystroke;
var EFSpecial_KeystrokeEx=AFSpecial_KeystrokeEx;
function EFSpecial_Format(){
  event.target.savevalue=event.value;
```

```
  event.target.ghBuiltin=true;
  AFSpecial_Format.apply(null,arguments);
}
var EFRange_Validate=AFRange_Validate;
var EFSimple_Calculate=AFSimple_Calculate;
var EFMergeChange=AFMergeChange;
\end{insDLJS}
```

`\nocalcs` is `\relax` unless the `nocalcs` option is taken, in which case it is `\let` to `\endinput`.

```
\nocalcs
\begin{insDLJS}{ghsupport}{gh: Support for the Calculate Event}
```

In order to get the gray hints to appear in the terminal field of a calculation group, we cannot perform the calculate when all the dependent fields are empty. `cArray` is an array of all dependent fields involved in the calculation. The use of this function is illustrated in `gh-eforms.tex` and `gh-hyperref.tex`.

```
function AllowCalc(cArray) {
  var f,g;
  for (var i=0; i<cArray.length; i++) {
    f=this.getField(cArray[i]);
    g=f.getArray();
    for (var j=0; j<g.length; j++)
      if (g[j].valueAsString!="") return true;
  }
  return false;
}
\end{insDLJS}
```

```
%</package>
```

3 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

4 Change History

v1.1 (2018/10/04)

General: Deprecated the usealtadobe option, now automatically loaded 2

Modify \FmtToGray, \FocusToBlack, and \BlurToBlack for better behavior when entry is left empty 1

v1.2 (2018/11/01)

General: Added \CommitSuccessEvent 5

Added \EnterCommitFailDef 4

Added \FailStringDef 4

Revisions to support PDF-XChange Editor . . . 1