AcroTEX.Net

richtext: A method of creating rich text strings

D. P. Story

Prepared: July 2, 2020
www.acrotex.net
Version 1.1.1, 2020/07/02

Table of Contents

1 Introduction **3**

2 Preamble: Required packages and options **3**

3 Creating rich text strings **3**

3.1 The Font and Link tabs 7

- The Font tab 7
- The Link tab 10
- Miscellaneous markup of the Font classification 10

¶ Bold and italic 10

¶ Subscripts and superscripts 11

3.2 The Paragraph tab 11

- Miscellaneous markup for the Paragraph classification . . . 16

¶ Starting a new line using \br 16

¶ Adding a space with \spc 17

¶ Using the raw key 17

¶ Special characters 18

4 Rich text fields **18**

4.1 The **DS** key 19

4.2 The **RV** and **V** keys 20

- Single paragraph fields 20
- Multiple paragraph fields 21

5 Displaying the values of the RV and V keys **23**

References **26**

1. Introduction

Rich text contents for variable text (text fields and editable combo boxes) and markup annotations was introduced into the PDF specification beginning with PDF 1.5 (Acrobat and Adobe Reader version 6). The rich text strings are difficult to create for it requires reading from a number of sources. The richtext package provides commands and documentation needed to "easily" produce such rich strings. We demonstrate the results using the eforms package (the text field produced by hyperref does not support rich text).

References for this material includes the *PDF Reference* [4], the XFA specification [1], and the CSS2 specification [2]. Additionally, the *JavaScript for Acrobat API Reference* [3] covers the JavaScript API for handling rich text content.

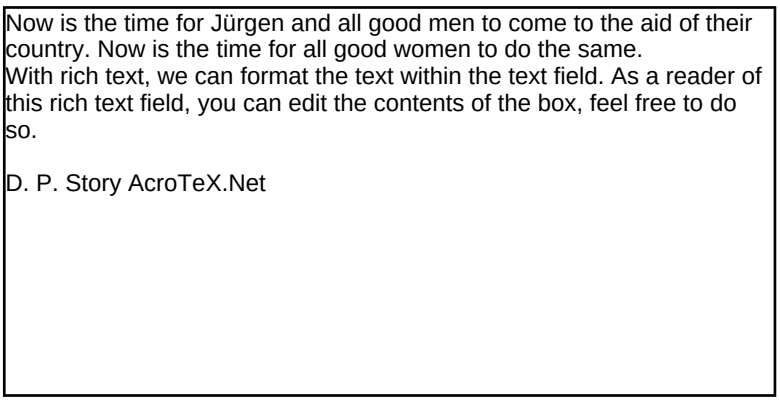
2. Preamble: Required packages and options

The package has no options and only requires xkeyval and ifxetex packages. The package can produce rich text strings, but to actually use them, you'll need the eforms package.

The package works for all drivers dvips, pdflatex, xelatex, and luatex. The eforms package can automatically detect all drivers except dvips, and that is used by default.

3. Creating rich text strings

We begin by illustrating the result of the richtext package, consider the rich text field below.



To edit the field above, click in the field, press Ctrl+E or Cmd+E (for Mac OS) to obtain the Form Field Text Properties toolbar. By pressing the More button, you can see the additional properties of the field, as seen in Figure 1.

A rich text may have any of several style attributes, many of these are illustrated in the above example. As a guide to introducing the attributes, we follow the Form Field Text Properties dialog box shown in Figure 1.

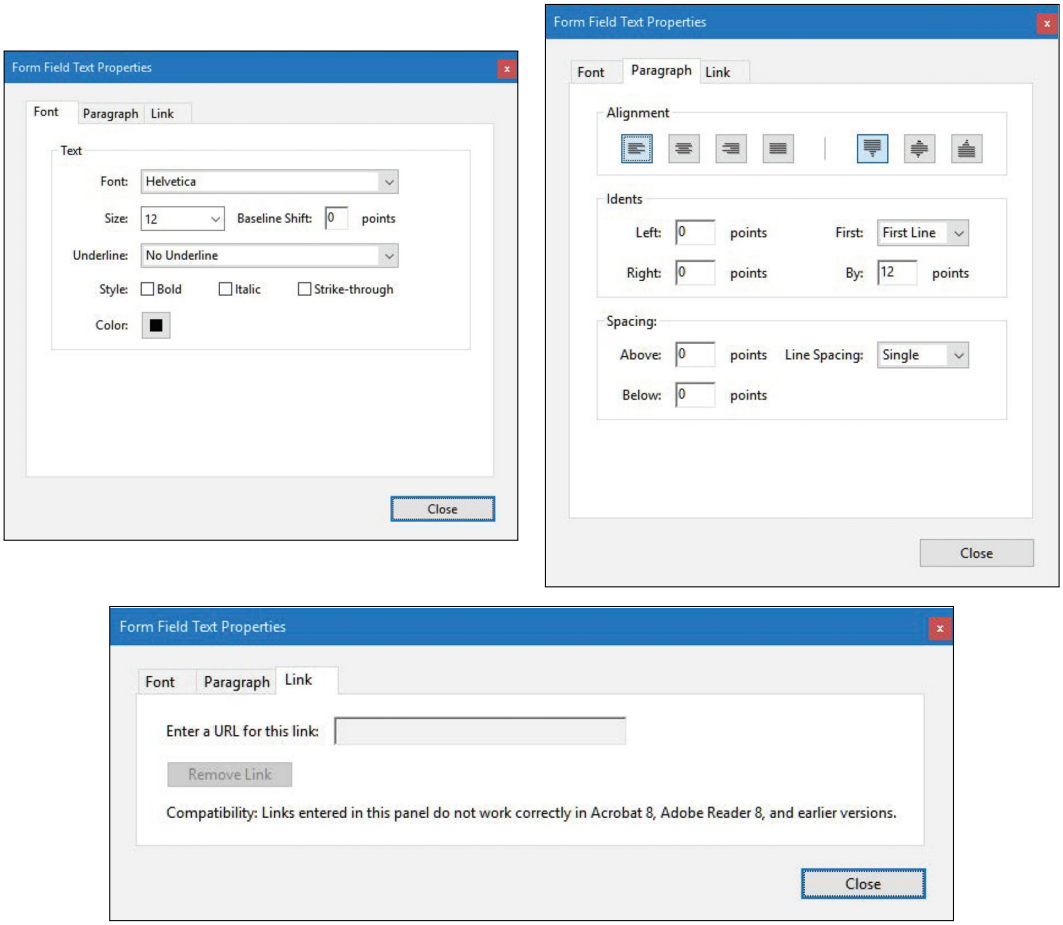


Figure 1: The Font, Paragraph and Link tabs

The basic command for creating a rich text *paragraph* is `\rtpara`:

`\rtpara[⟨Para-Font-attrs⟩]{⟨name⟩}{⟨rich-text-paragraph⟩}`

where ⟨*Para-Font-attrs*⟩ are key-value pairs (in the LaTeX sense) that are described in Sections 3.1 and 3.2; these attributes are applied to the paragraph as a whole. The ⟨*name*⟩ is a unique name to be associated with ⟨*rich-text-paragraph*⟩ so it can be referenced later from within a text field. There are two types of attributes: Font and Paragraph, as guided by Figure 1. For convenience, the Link attributes (URLs) are classified as Font. The optional argument of `\rtpara` consists of usually Paragraph attributes, most Font attributes are also recognized.

Important! These comments are focused on the dvips -> distiller workflow. The dvips application when converting from DVI to PS inserts line breaks in the Postscript file, according to rules known only to the developer. Breaking lines can cause problems with JavaScript code and rich text markup. The problems encountered do not occur in the other workflows (pdflatex, lualatex, and xelatex).

dvips->distiller workflow

Rule 1. For the workflow dvips -> distiller, spaces in the third argument (⟨*rich-text-paragraph*⟩) are obeyed and are replaced with the octal `\040` which is the (PDF) space character. Of particular importance, the third argument of `\rtpara` must be flush against the left margin, otherwise spurious spaces are introduced. See the verbatim listing below.

dvips->distiller workflow

Rule 2. Within the argument of the `\span` command, *do not use spaces*. To break a line within the argument of `\span`, use the comment character (%) to avoid spurious spaces, see line ❶ below. The *value* of a key may require spaces, for example, `\span{font='Myriad Pro'}`. There should be no spaces between key-value pairs, for example, don't type something like this: `\span{font = 'Myriad Pro'}`, or do this: `\span{style={bold, italic, strikeit}`.

The definition of the first paragraph of the above rich text field reads as follows:

```
\rtpara[indent=first]{para1}                % this is OK
{Now is the time for
\span{style={bold,italic,strikeit},%        ❶
color=ff0000}{J\374rgen}
and all good men to come to the aid of \it{their}
\bf{country}. Now is the time for \span{style=italic}
{all good} women to do the same.}
```

In this example, the optional argument for \rtpara was used to indent the paragraph. The rich text defined here is named para1. The third argument, ⟨*rich-text-string*⟩, consists of ordinary text, the \span command used to insert special formatting for text, and certain other 'short-cut' markups like \it and \bf. Note that the umluat (ü) is expressed as octal (\374).

The \span command is used to format individual sentence fragments. Its syntax is,

```
\span[⟨Font-attrs⟩]{⟨rich-text-string⟩}
```

where ⟨*Font-attrs*⟩ are Font attributes as described in Sections 3.1; these attributes are applied to the string ⟨*rich-text-string*⟩ only. The \span command, as described here, is only defined within the third argument (⟨*rich-text-paragraph*⟩) of \rtpara. This is necessary because \span is a TEX primitive command, and we must not overwrite its definition.

When you create a *rich text string* there is a parallel development of a *plain text string*, the string without its rich text markup, these two (rich and plain strings) are used to populate the values of the **RV** and **V** keys of a text field. When you define a rich text paragraph string under its own ⟨*name*⟩, you can typeset it (to check the syntax) and its plain text counterpart using the methods of Section 5. For example,

```
\useRV{para1}: <p dir="ltr" style="text-indent:12pt;
margin-top:0pt;margin-bottom:0pt;">Now is the time for
<span style="text-decoration:line-through;font-weight:bold;
font-style:italic;color:#ff0000;">J\374rgen</span> and all
good men to come to the aid of <i>their</i> <b>country</b>.
Now is the time for <span style="font-style:italic;">all
good</span> women to do the same.</p>

\useV{para1}: Now is the time for J\374rgen and all good
men to come to the aid of their country. Now is the time
for all good women to do the same.
```

The commands \useRV{⟨*name*⟩} and \useV{⟨*name*⟩} may also be used to insert the strings into the **RV** and **V** keys, respectively; though the richtext package offers an alternative technique.

3.1. The Font and Link tabs

In this section, we cover the Font and Link tabs, as well as other attributes not listed on any tab.

• The Font tab

We discuss the Font tab of Figure 1. The key-value for each of the attributes is given and described briefly. These key-values may appear as ⟨*Font-attrs*⟩ or ⟨*Para-Font-attrs*⟩.

Font: `font=`⟨*font_name*⟩ A font name or a list of font names to be used to display the enclosed text. The first entry is the font name of the font to use. The second font name is typically a generic family name to use if an exact match is not found. The generic family names are `symbol`, `serif`, `sans-serif`, `cursive`, `monospace`, and `fantasy`. The default is `sans-serif`. If a typeface name contains white space, enclose it within single quotes (').

```
\rtpara[font={Arial,sans-serif}]{para1}{This is Arial or a
san-serif substitute.}
```

This is Arial or a san-serif substitute.

```
\rtpara{para2}{This is \span{font='Myriad Pro'}
{Myriad Pro} font.}
```

This is Myriad Pro font.

In the second example, only 'Myriad Pro' is actually set in the Myriad Pro font; the rest of the sentence is typeset in the default font, Helvetica in this case. Use Ctrl+E (Cmd+E) to inspect the properties of these two fields and verify the fonts are Arial, Myriad Pro, and Helvetica.

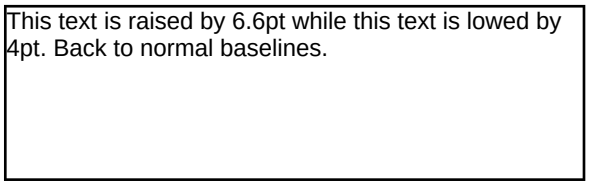
Size: `size=`⟨*dec_num*⟩ The size of the font to be used. The value of `size` is ⟨*dec_num*⟩, a (positive) decimal number.

```
\rtpara[size=12]{para1}{This is 12pt font, while
   \span{size=8}{this is 8pt font.} OK?}
```

This is 12pt font, while this is 8pt font. OK?

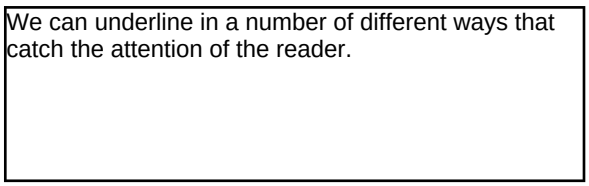
Baseline Shift: `raise=⟨def_num⟩` The position of the baseline of the text is determined by the `raise` key. `raise=6.6` raises the baseline `6.6pt`, while `raise=-4` lowers it `4pt`.

```
\rtpara{para1}{This text \span{raise=6.6}{is raised by
    6.6pt} while this text \span{raise=-4}
   {is lowed by 4pt.} Back to normal baselines.}
```



Underline: `ulstyle=⟨none|ul|2ul|wul|2wul⟩` The `ulstyle` key determines the style of underlining, possible values are `none` (no underlining), `ul` (underlining), `2ul` (double-line underlining), `wul` (word underlining), and `2wul` (double-line word underlining).

```
\rtpara{para1}{We can \span{ulstyle=ul}{underline in a}
   \span{ulstyle=2ul}{number of different ways}
   \span{ulstyle=wul}{that catch the}
   \span{ulstyle=2wul}{attention of the reader}.
```



Style: `style={[bold,][italic,][strikeit]}` Unlike some of the other (choice) keys, the value of the `style` key is any *subset* of the values listed: for example, `style=bold` paints the underlying text in bold, `style={bold,italic}` yields bold-italic font, and, for a final example, `style={italic,strikeit}` typesets its text in strike-through italic. Multiple values must be enclosed in braces ({}) so that `xkeyval` can correctly parse them.

```
\rtpara{para1}{To \span{style=bold}{boldly to go} where
   \span{style={bold,italic}}{no man has gone}
   \span{style={italic,strikeit}}{prior}
   \span{style={italic,bold}}{before.}
```

To boldly to go where no man has gone priorbefore.

Color: color=⟨*rrggbb*|{rgb(*rrr,ggg,bbb*)}⟩ Use this key to color the effected text. There are two methods of defining color:

(1) *rrggbb* uses a 2-digit hexadecimal value for each component;

(2) rgb(*rrr,ggg,bbb*) uses a decimal value (0–255) for each component.

Because the second form contains commas, it must necessarily be enclosed in braces ({}) to be correctly parsed by xkeyval.

```
\rtpara{para1}{This is \span{color={rgb(255,0,0)}}{red} and
   this is \span{color=0000ff}{blue}.
```

This is red and this is blue.

Things are not as bad as it seems. The xcolor package has the wonderful command \convertcolorspec that converts colors between color models. For example, we might define:

```
\convertcolorspec{named}{red}{RGB}{\rgbRed}
\convertcolorspec{named}{blue}{HTML}{\htmlBlue}
\convertcolorspec{named}{magenta}{RGB}{\rgbMagenta}
\convertcolorspec{named}{magenta}{HTML}{\htmlMagenta}
```

We can then use these named colors.

```
\rtpara{para1}{This is \span{color={rgb(\rgbRed)}}{red} and
this is \span{color=\htmlBlue}{blue}. We can do magenta two
ways, using \span{color={rgb(\rgbMagenta)}}{decimal
components} or using \span{color=\htmlMagenta}{hexadecimal
components}.}
```

This is red and this is blue. We can do magenta two ways, using decimal components or using hexadecimal components.

Notice that color={rgb(\rgbMagenta)}, the value of color, is still enclosed in braces since the expansion of \rgbMagenta contains commas.

• The Link tab

We can create a link within rich text by using the `url` key from within the first argument of the `\span` command. The syntax is `url=`⟨*URL*⟩.

```
\rtpara{para1}{Visit me at \span{url={http://www.acrotex.net},%
font='Courier New'}{http://www.acrotex.net}}
```

Visit me at http://www.acrotex.net

It appears the Acrobat/Reader applications format a URL in underlined blue. We can override this however.

```
\rtpara{para1}{Visit me at \span{url={http://www.acrotex.net},%
color=\htmlMagenta,ulstyle=none,font='Courier New'}%
{http://www.acrotex.net}}
```

Visit me at http://www.acrotex.net

Special characters are no problem, with the exception of wrapping a long URL around to a different line (usually needed for display purposes):

```
\rtpara{para1}{Visit me at
\span{url={http://www.math.uakron.edu/~dpstory/%
acrotex.html#technical}}{AcroTeX at The University
of Akron}}
```

Visit me at AcroTeX at The University of Akron

• Miscellaneous markup of the Font classification

There are several other attributes that are not key-values, but are implemented through LaTeX commands.

¶ Bold and italic. There are a couple of XHTML elements that can also be used for bold and italic.

- `\bf{`⟨*text*⟩`}` expands to `<b>`⟨*text*⟩`</b>` and places ⟨*text*⟩ in bold font. May be used within a `\span` command.
- `\it{`⟨*text*⟩`}` expands to `<i>`⟨*text*⟩`</i>` and places ⟨*text*⟩ in italic font. May be used within a `\span` command.

Both \bf and \it are local commands, undefined outside of the third argument of \rtpara. Do not code ⟨*text*⟩ or ⟨*text*⟩ directly, rather, always use the LaTeX commands \bf and \it. \bf and \it may be nested.

```
\rtpara{para1}{We \bf{boldly} say that \it{italic} is used
for emphasis, but both \span{color=\htmlBlue}%
{\bf{\it{drive home the point}}}.}
```

We boldly say that italic is used for emphasis, but both drive home the point.

¶ Subscripts and superscripts. Subscripts and superscripts are implemented through LaTeX commands \sub and \sup.

- \sub{⟨*text*⟩} expands to _{⟨*text*⟩} and places ⟨*text*⟩ as a subscript.
- \sup{⟨*text*⟩} expands to ^{⟨*text*⟩} and places ⟨*text*⟩ as a superscript.

Both \sub and \sup are local commands, undefined outside of the third argument of \rtpara. Do not code these raw markups, rather always use \sub and \sup.

```
\rtpara{para1}{When we compile $x_2^3$ we get
\it{x}\sub{2}\sup{3}, nicely typeset or would you prefer
\it{x}\sup{3}\sub{2}?}
```

When we compile x_2^3 we get x23, nicely typeset or would you prefer x32?

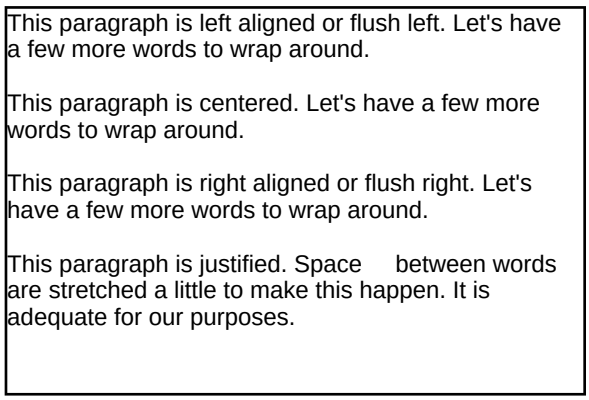
3.2. The Paragraph tab

We begin by following the Paragraph tab of Figure 1. The top-most region on the Paragraph tab is labeled Alignment. It consists of two separated regions, the one on the left is *Horizontal Alignment*, the one on the right is *Vertical Alignment*.

Alignment:

Horizontal Alignment: halign=⟨left|center|right|justify⟩
The meaning of these key-values are obvious, we'll illustrate with examples.

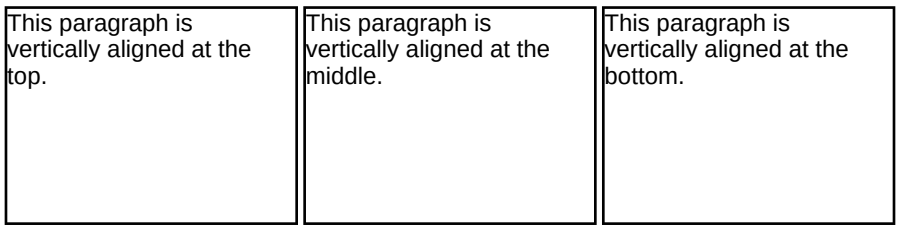
```
\rtpara[halign=left]{para1}{This paragraph is left
aligned or flush left. Let's have a few more words
to wrap around.}
\rtpara[halign=center]{para2}{This paragraph is
centered. Let's have a few more words to wrap
around.}
\rtpara[halign=right]{para3}{This paragraph is right
aligned or flush right. Let's have a few more words
to wrap around.}
\rtpara[halign=justify]{para4}{This paragraph is
justified. Space between words are stretched a
little to make this happen. It is adequate for
our purposes.}
```



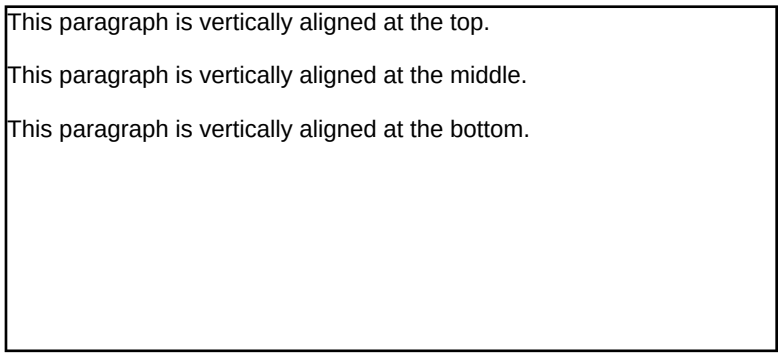
Horizontal alignment is applied to individual paragraph, unlike vertical alignment.

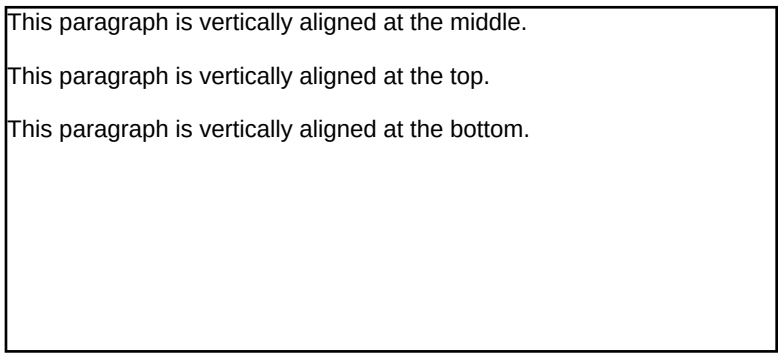
Vertical Alignment: valign=⟨top|middle|bottom⟩ Again, we shall illustrate by example.

```
\rtpara[valign=top]{para1}{This paragraph is vertically
aligned at the top.}
\rtpara[valign=middle]{para2}{This paragraph is
vertically aligned at the middle.}
\rtpara[valign=bottom]{para3}{This paragraph is
vertically aligned at the bottom.}
```



The `valign` key seems to apply to all paragraphs in the rich text form field, as illustrated below.





The vertical alignment for the whole rich text field obeys the `valign` key of the first paragraph.

Indents: Through the Indents region of the Paragraph tab, left and right margins may be set, as well as the amount of indent.

Left: `margleft={dec}` The value of `{dec}` is a nonnegative decimal number, it represents the number of points to make the left margin.

Right: `margright={dec}` The value of `{dec}` is a nonnegative decimal number, it represents the number of points to make the right margin.

Below is an example for both `margleft` and `margright`.

```
\rtpara[margleft=10,margright=40,halign=justify]%
{para1}{This is the first paragraph, it has
a left margin of 10pt and a right margin of
40pt.}
\rtpara[halign=justify]{para2}{This is the second
paragraph. We demonstrate that the left and
margins can be applied separately to
paragraphs.}
```

This is the first paragraph, it has a left margin of 10pt and a right margin of 40pt.

This is the second paragraph. We demonstrate that the left and margins can be applied separately to paragraphs.

First & By: Two key-values: `indent=⟨none|first|hanging⟩` & `indentby=⟨`*dec*`⟩` When `indent` key is set to `indent=first`, the first line is indented by an amount of ⟨*dec*⟩`pt`; similarly, if `indent=hanging`, there is a hang indent on the first line by an amount of `-`⟨*dec*⟩`pt` (the minus sign (`-`) is automatically applied. The default indent amount it `12pt`.

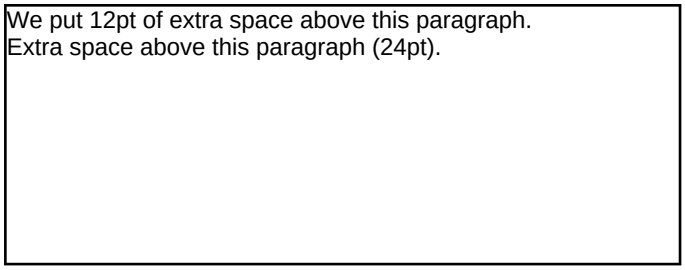
```
\rtpara[indent=first]{para1}{This paragraph is
indented by the default amount of 12pt.}
\rtpara[indent=first,indentby=24]{para2}{In this
second paragraph, we indent by 24pt, twice
as wide as the default.}
\rtpara[indent=hanging]{par3}{Here we have a third
paragraph, separated from the other two, with
the default hanging indentation.}
```

This paragraph is indented by the default amount of 12pt.
In this second paragraph, we indent by 24pt, twice as wide as the default.

Here we have a third paragraph, separated from the other two, with the default hanging indentation.

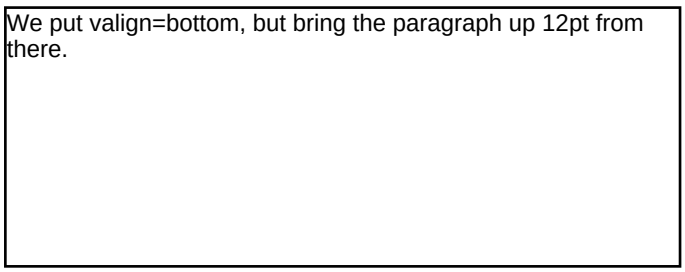
Spacing: Above: `margtop=⟨dec⟩` A value of ⟨*dec*⟩ (positive, negative, or zero) adds vertical space *above* the paragraph.

```
\rtpara[margtop=12]{para1}{We put 12pt of extra
space above this paragraph.}
\rtpara[margtop=24]{para2}{Extra space above this
paragraph (24pt).}
```



Below: `margbottom=⟨dec⟩` A value of ⟨*dec*⟩ (positive, negative, or zero) adds vertical space below the paragraph.

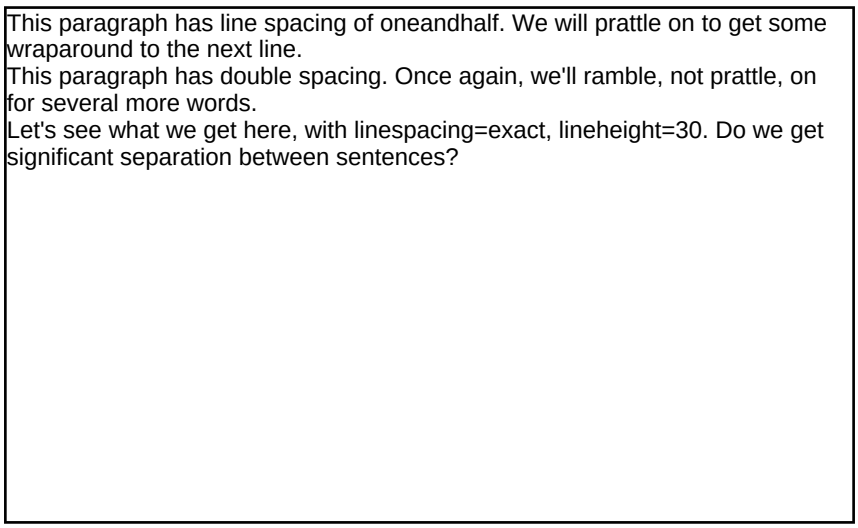
```
\rtpara[valign=bottom,margbottom=12]{para1}{We put
\span{font=Courier,style=bold}{valign=bottom},
but bring the paragraph up 12pt from there.}
```



Line Spacing Sets the amount of vertical space between baselines. The key-values are

`linespacing=⟨single|oneandhalf|double|exact⟩`

When `linespacing=exact`, use `lineheight=⟨dec⟩` to set the space between baselines.



The paragraph declarations for the above rich text field are,

```
\rtpara[linespacing=oneandhalf]{para1}{This
paragraph has line spacing of oneandhalf. We
will prattle on to get some wraparound to the
next line.}
\rtpara[linespacing=double]{para2}{This paragraph
has double spacing. Once again, we'll ramble,
not prattle, on for several more words.}
\rtpara[linespacing=exact,lineheight=30]{para3}
{Let's see what we get here, with
linespacing=exact, lineheight=30. Do we
get significant separation between sentences?}
```

The value of `lineheight`, which gives a 'squeezing' effect between lines of the paragraph.

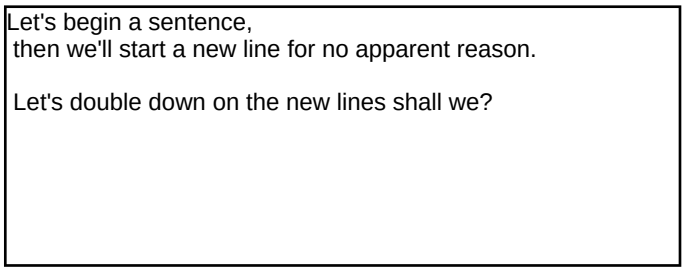
• Miscellaneous markup for the Paragraph classification

There are several other features that do not fit conveniently anywhere else, so here they are.

¶ Starting a new line using \br. The `\br` command expands to `<br />`. It should not be put within the second argument of the `\span` command. As was the case with `\bf`, `\it`, `\sub`, and `\sup`, do not code in `<br />` directly, for you will fail.

```
\rtpara{para1}{Let's begin a sentence,\br then we'll
start a new line for no apparent reason.\br\br
```

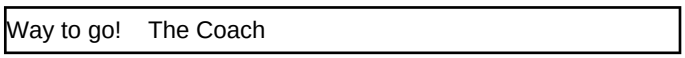
```
Let's double down on the new lines shall we?}
```



¶ Adding spaces with \spc. As with TEX multiple spaces are ignored. To insert additional 'hard' spaces into the data stream, use the `\spc` command. (This is a local command that is undefined outside `\rtpara`.

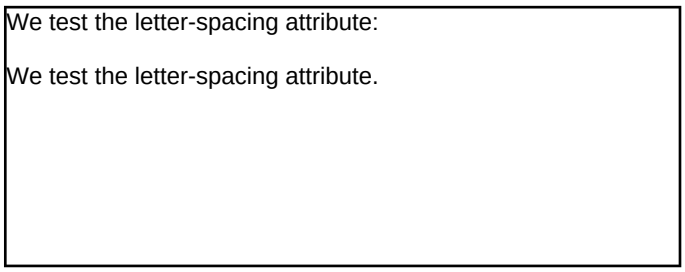
```
\rtpara{para1}{Way to go!\spc\spc\spc\spc The Coach}
```

Here we induce four hard spaces.



¶ Using the raw key. There is another key, the `raw` key, that can be used within the optional argument of `\rtpara` or within the first argument of `\span`. Using this key, you can pass raw CSS2 markup.

```
\rtpara{para1}{We test the letter-spacing
attribute:\br\br\span{raw=letter-spacing:0.25em;}
{We test the letter-spacing attribute.}}
```



The syntax for a CSS2 attribute is '⟨*key*⟩:⟨*value*⟩;', that is, keys and values are separated by a colon (`:`) and the value is terminated with a semicolon (`;`).

It appears that tab stops work as well, these can be specified using the `raw` key as well. Refer to the XFA Specifications [1].

¶ Special characters. The richtext handles special characters pretty well. Before \rtpara reads its third argument (⟨*rich-text-paragraph*⟩), a number of changes in \catcodes and redefinitions occur. Within \rtpara, the following characters *do not need to be* escaped: $, #, and ~ (tilde). The following characters *need to be* escaped: \<, \>, \&, \% (the comment character (%) retains its LaTeX meaning), \{, and \} (the left and right braces have their usual TeX/LaTeX meaning). The single quote (') and double quote (") may be optionally escaped (to \' and \"). Escape them if something goes wrong. Use the command \cs{⟨*text*⟩} to obtain a literal backslash ('\'); for example \cs{LaTeX}, shown below, expands to '\LaTeX'.

```
\rtpara{para1}{We \"test\" \'special\' \bf{characters:}
\<\>\&\{ #\% in \cs{LaTeX} $x^2_4$ becomes
\it{x}\sup{2}\sub{4} \{\}}
```

The above \rtpara paragraph has two forms the **RV** form and the **V** form; these can be seen by using the \useRV and \useV commands.

\useRV{para1}:
```
<p dir="ltr" style="margin-top:0pt;
margin-bottom:0pt;">We &quot;test&quot; 'special'
<b>characters:</b> &lt;&gt;&amp;\{ #% in \\LaTeX $x^2_4$
becomes <i>x</i><sup>2</sup><sub>4</sub> \{\}</p>
```

\useV{para1}:
```
We "test" 'special' characters: <>&\{ #% in
\134LaTeX $x^2_4$ becomes x24 \{\}
```

The resulting rich text form field is seen below:

We "test" 'special' characters: <>&{ #% in \LaTeX x^2_4 becomes x24 {}

That's pretty cool!

4. Rich text fields

Up to this point in the manual, the discussion has focused on creating rich text strings. They may be fun to create and look at, but usually we want to insert them into a text field. The comments here are for eforms package, having checked with hyperref to see if there is a **RV** key, there is not.

To create a rich text field, use the \textField command of eforms:

```
\textField[\Ff{\FfRichText}\Ff{\FfMultiline}
    \DS{⟨defaultstyle⟩}\RV{⟨rich-value⟩}\V{⟨value⟩}
]{⟨fld-name⟩}{⟨width⟩}{⟨height⟩}
```

Remove \Ff{\FfMultiline} if the field is only a single line. We discuss the **DS** key (\DS) key first, followed by the keys **RV** and **V** (\RV and \V).

4.1. The DS key

The value of the **DS** key sets the formatting for the text field as a whole. Most importantly, use it to set the font, text size, and color. There is a built-in default style, defined below:

```
\newcommand\useDefaultDS{font-family:Helvetica,sans-serif;
    font-size:12.0pt;font-style:normal;font-weight:normal;
    text-align:left;color:#000000}
```

You may redefine it to suit your purposes, but this is what Acrobat/Adobe Reader sets as the default style. I would recommend \setDefaultStyle to define your own custom default style. \useDefaultDS is the reason why most all rich text fields in this document use Helvetica at 12pt! Use \useDefaultDS as follows, shown in bold font:

```
\textField[\Ff{\FfRichText}\Ff{\FfMultiline}
    \DS{\useDefaultDS}\RV{⟨rich-value⟩}\V{⟨value⟩}
]{⟨fld-name⟩}{⟨width⟩}{⟨height⟩}
```

To create a custom default style use \setDefaultStyle.

```
\setDefaultStyle{⟨name⟩}{⟨Font-Para-attrs⟩}
\useDS{⟨name⟩}
```

Typically, the key-values associated with the Font tab, Section 3.1, may be used, some key-values are removed, such as ul, raise, and url. When you've defined a custom default style using \setDefaultStyle, insert \useDS{⟨name⟩} as the value of the **DS** key.

```
\rtpara{para1}{The font should be \'Myriad Pro\' at 10pt
and the default color of the field is webbrown, a color
defined in the web package.}
```

```
\setDefaultStyle{myStyle}{font='Myriad Pro',size=10,
color=990000}
\textField[\Ff{\FfRichText}\Ff{\FfMultiline}
  \DS{\useDS{myStyle}}\RV{\useRV{para1}}\V{\useV{para1}}
]{rtFld30}{3in}{16bp*3}
```

The font should be 'Myriad Pro' at 10pt and the default color of the field is webbrown, a color defined in the web package.

Note the use of \useRV and \useV in the **RV** and **V** fields. These are discussed in the next section.

4.2. The RV and V keys

The techniques to handle multiple paragraph fields are more complex (but not discouragingly so), that topic will be taken up after the discussion of single paragraph fields.

• Single paragraph fields

For a single paragraph field, there is only one \repara defined prior to the field. This string data (both rich and plain) are inserted into the \RV and \V keys using \useRV and \useV. We repeat the previous example, but with the emphasis on \RV and \V, and not on \DS.

```
\rtpara{para1}{The font should be \'Myriad Pro\' at 10pt
and the default color of the field is webbrown, a
color defined in the web package.}
\setDefaultStyle{myStyle}{font='Myriad Pro',size=10,
color=990000}
\textField[\Ff{\FfRichText}\Ff{\FfMultiline}
  \DS{\useDS{myStyle}}\RV{\useRV{para1}}\V{\useV{para1}}
]{rtFld30}{3in}{16bp*3}
```

The font should be 'Myriad Pro' at 10pt and the default color of the field is webbrown, a color defined in the web package.

We declare our rich paragraph string using \rtpara and name it para1. We insert two data streams, one into the rich text key (\RV{\useRV{para}}) and the other into the (plain) text key (\V{\useV{para}}).

• Multiple paragraph fields

The strategy is to use several `\rtpara` commands to declare several rich text paragraph. What is the best way to 'paste' these paragraphs together? The method developed is to use `\setRVVContent` command.

`\setRVVContent{⟨name⟩}{ {⟨name1⟩}{⟨name2⟩}...{⟨namek⟩} }`

where ⟨*name*$_i$⟩ is the name of a rich text paragraph string, or is the keyword `skipline`. The keyword `skipline` is case-sensitive, it must be typed exactly. The role `skipline` plays is to insert a blank line between paragraphs; `{skipline}` inserts one blank line between paragraphs.

Having composed how the strings are to be put together, we need to insert them into **RV** and **V**.

`\useRVContent{⟨name⟩}`
`\useVContent{⟨name⟩}`

where ⟨*name*⟩ is the name given in a previous `\setRVVContent` command. Insert `\useRVContent` as the value of the **RV** key, and `\useVContent` as the value of the **V** key.

We take as an example, the one from Section 3.

```
\rtpara[indent=first]{para1}{Now is the time for
\span{style={bold,italic,strikeit},color=ff0000}{J\374rgen}
and all good men to come to the aid of \it{their}
\bf{country}. Now is the time for \span{style=italic}%
{all good} women to do the same.}
\rtpara[indent=first]{para2}{With rich text, we can format the
text within the text field. As a reader of this rich text
field, you can edit the contents of the box, feel free to
do so.}
\rtpara[halign=right]{para3}{D. P. Story
\span{url=http://www.acrotex.net}{AcroTeX.Net}}
```

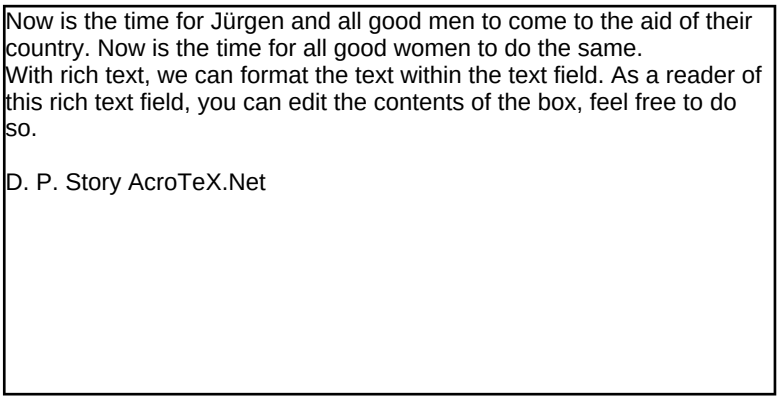
Now set the content with `\setRVVContent`, naming it `myContent`.

```
\setRVVContent{myContent}{{para1}{para2}{skipline}{para3}}
```

Having done all that, we create our rich text field:

```
\begin{center}
\textField[\Ff{\FfRichText}\Ff{\FfMultiline}
  \DS{\useDefaultDS}
  \RV{\useRVContent{myContent}}
  \V{\useVContent{myContent}}
]{rtFld31}{4in}{10\baselineskip}
\end{center}
```

where, the \RV and \V keys are highlighted in bold for your viewing pleasure. The rich text field the result of these declarations.



The argument of \setRVVContent is pretty robust. In making our declarations, we can type:

```
\setRVVContent{myContent}
{
    {para1}
    {para2}
    {skipline}
    {skipline}
    {para3}
}
```

Note that I've added a skipline so that are two blank lines after the second paragraph and before the third paragraph.

5. Displaying the values of the RV and V keys

For the purposes of inspection or discussion, you can *typeset* the rich text values for both the **RV** and **V** keys using the following set of environments and commands.

```
\begin{displayRtPara}{⟨name⟩}
\rtpara[⟨Para-Font-attrs⟩]{⟨name⟩}{⟨rich-text-paragraph⟩}
\end{displayRtPara}

\begin{displayRtPara*}{⟨name⟩}
\rtpara[⟨Para-Font-attrs⟩]{⟨name⟩}{⟨rich-text-paragraph⟩}
\end{displayRtPara*}

\displayRV{⟨name⟩}
\displayV{⟨name⟩}
```

By placing an `\rtpara` command within the `displayRtPara` environment, you can then typeset a (cleaned-up) version of **RV** and **V** by expanding `\displayRV`⟨*name*⟩ and/or `\displayRV`⟨*name*⟩. The `displayRtPara*` environments produces the 'ugly' version of the entries, useful only with the dvips -> distiller workflow.

Example. If we say,

```
\begin{displayRtPara}{para1}
\rtpara[indent=first]{para1}{Now is the time for
\span{style={bold,italic,strikeit},color=ff0000}{J\374rgen}
and all good men to come to the aid of \it{their}
\bf{country}. Now is the time for \span{style=italic}%
{all good} women to do the same.}
\end{displayRtPara}
```

then type,

```
\begin{flushleft}\textbf{Rich text: }\ttfamily
\displayRV{para1}\end{flushleft}
\begin{flushleft}\textbf{Plain text: }\ttfamily
\displayV{para1}\end{flushleft}
```

The following is typeset into the document:

Rich text: Now is the time for J\374rgen and all good men to come to the aid of their country. Now is the time for all good women to do the same.

Plain text: Now is the time for J\374rgen and all good men to come to the aid of their country. Now is the time for all good women to do the same.

Now, if the same `\rtpara` is enclosed in a `displayRtPara` environment (not shown), we get the following:

Rich text: Now\040is\040the\040time\040for\040J\374rgen\040and\040all\040good\040men\040to\040come\040to\040the\040aid\040of\040their\040country.\040Now\040is\040the\040time\040for\040all\040good\040women\040to\040do\040the\040same.

Plain text: Now\040is\040the\040time\040for\040J\374rgen\040and\040all\040good\040men\040to\040come\040to\040the\040aid\040of\040their\040country.\040Now\040is\040the\040time\040for\040all\040good\040women\040to\040do\040the\040same.

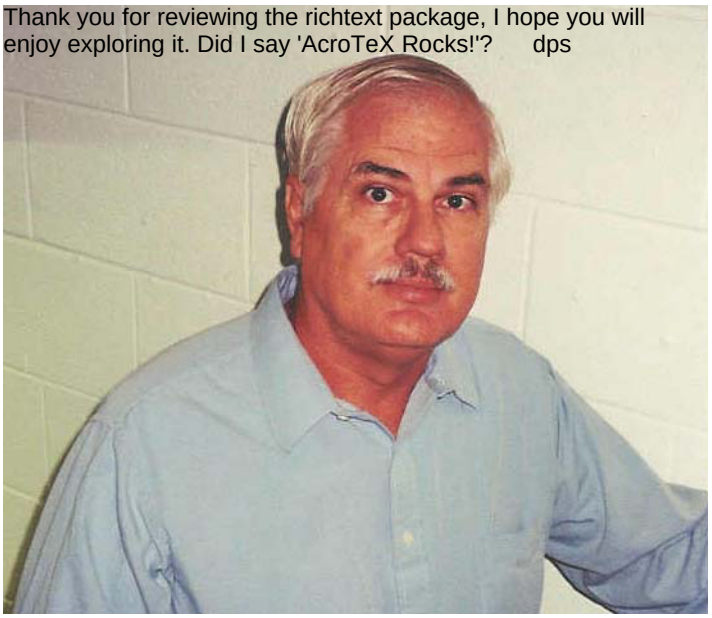
Yes, this document was created by a dvips -> distiller workflow![1] As you can see, `\040` is inserted whenever there is a space. This is to keep dvips from breaking the Postscript at a point that will break the rich text CSS2 markup.

For a non-Postscript workflow, typeset print outs from `displayRtPara` and `displayRtPara*` are the same, because only dvips needs special handling.

[1] Actually, the dvipsone -> distiller workflow.

That's about it! Now, back to my retirement. ☺

Did I say that you can write captions to figures using rich text?



References

[1] Adobe XML Forms Architecture (XFA) Specification, Version 3.3, Adobe Systems, Inc., Jan. 2012
https://reference.pdfa.org/iso/32000/wp-content/uploads/2017/04/XFA-3_3.pdf

[2] Cascading Style Sheets (CSS 2.2) Specification, Editors: Bert Bos *et al.*, World Wide Web Consortium (W3C), June 2011
https://www.w3.org/TR/CSS2/

[3] JavaScript for Acrobat API Reference, Adobe Systems, Inc., May 2015
https://www.adobe.com/devnet/acrobat/documentation.html

[4] PDF Reference, Sixth Edition, Version 1.7, Adobe Systems, Inc., 2006
https://www.adobe.com/devnet/pdf/pdf_reference_archive.html