



AcroTeX.Net

AcroTeX PDF Blog

Processing Acrobat Forms using JavaScript

External Processing of a Field

Part 2: Text Fields

D. P. Story

3. Text Fields

A *text field* is a rectangular region in which the user can enter text from the keyboard. A text field may be a single line, or may span multiple lines. In this article we explore methods of processing a text field, using a script, external to the field itself.

3.1. Getting and Setting the Value of a Text Field

We can get and set the value of a text field with the *Field.value* property.

• Getting the Value of a Text Field

To obtain the value of the text field, we first acquire the *Field* object of the target field, then use *Field.value*.

Example 3.1. From a button, acquire the current value of a text field and display the result in an alert box.

Enter text into the field:

The above field accepts Rich Text Formatting; for simple text styles, right click the text field and select Text Style from the context menu, or press **Ctrl+E** to get the Properties toolbar. The *Field.value* property returns the *unformatted value* of the text field. See [Example 3.2](#) for an example of getting rich text.

The verbatim listing of the JavaScript executed off a Mouse Up trigger of the push button is given below:

```
1 var f = this.getField("myTxt3-1");
2 var v = f.value;
3 var stripv = v.replace(/\s*/g,"");
4 if ( stripv == "" )
5     app.alert("Nothing but white space in the field.");
6 else
7     app.alert("You entered \"" + v + "\" into the text field!");
```

Comments. Get the *Field* object of the target field (1); get the value of the target field (2) using the *value* property of the *Field* object; use a regular expression to strip out all white space (3) and save the result as *stripv*; (4) if *stripv* is an empty string, the user is trying to fool us, we let him have it (5), otherwise we report the result (7). □

Rich Text Field. When the text field is set to take Rich Text Formatting, and you want to somehow process the formatting, use *Field.richValue* to get and set the value of a rich

text field. According the *JavaScript for Acrobat API Reference*, the value of an array of Span objects. Refer to that reference for details on the Span object.

Example 3.2. Enter text into the field:

The above field accepts Rich Text Formatting; for simple text styles, right click the text field and select Text Style from the context menu, or press Ctrl+E to get the Properties toolbar. The *Field.value* property returns the *unformatted value* of the text field.

When you enter a value in the text field, the message that appears in the alert dialog box is must unsatisfactory. The value of *Field.richValue* is an array of Span objects. Obviously, dialog boxes, like the alert, do not support rich text formatting.

The verbatim listing of the JavaScript executed off a Mouse Up trigger of the push button is given below:

```
1 var f = this.getField("myTxt3-2");
2 var v = f.value;
3 var stripv = v.replace(/\s*/g,"");
4 if ( stripv == "" )
5     app.alert("Nothing but white space in the field.");
6 else {
7     v = ( f.richText ) ? f.richValue : f.value;
8     app.alert("You entered \"" + v + "\" into the text field!");
9 }
```

Comments. We use *Field.value* to determine if any non-white space has been entered into the text field. If the field is nonempty, we get the rich value (7) and report it in an alert. In line (7), we do a cool thing (using the condition operator of core JavaScript), we test whether the field is rich using the *Field.richText* property, and get the value of the field by using *Field.richValue* if the field accepts rich text formatting, and by using *Field.value*, otherwise. □

• Setting the Value of a Text Field

Here we explore setting the value of a text field.

Example 3.3. Get text from the user using a response dialog box, and populate this value in a text field.

The verbatim listing of the push button action follows.

```
1 var rtn = app.response({
2   cTitle: "AcroTeX PDF Blog",
3   cQuestion: "What do you think of the AcroTeX PDF Blog? "
4     + "Enter a response and be nice!"
5 });
6 if ( rtn != null ) {
7   var f = this.getField("myTxt3-3");
8   f.value = rtn;
9 }
```

Comments. In line (1) the `app.response` method is used to get the beloved user's thoughts and feelings, this method returns `null` if the user dismisses the dialog; we set the parameters of `app.response`, in (2) we wallow in blatant and shameless advertising, and (3) we pose the question; line (6), if the return value `rtn` is non-null we get the field object; and in line (7) we use `Field.value` to set the value of the field using the value `rtn` returned by `app.response`.

Documentation for the `app.response` method is found in the App object section of the [JavaScript for Acrobat API Reference](#). □

Rich Text Field. Let's have another go at processing a rich text field, but we need a field to place the rich value in

Example 3.4. Enter rich text into the field:

Transfer this value to this field:

In this version 8 of Acrobat/Adobe Reader, the rich text formatting was broken. This example should work for versions 6–7.x, and for version 9.¹

The verbatim listing of the push button action follows.

```
1 var f = this.getField("myTxtIn3-4");
2 var g = this.getField("myTxtOut3-4");
3 var v = f.value;
4 var stripv = v.replace(/\s*/g,"");
5 if ( stripv == "" )
6   app.alert("Nothing but white space in the field.");
7 else
8   g.richValue = f.richValue;
```

Comments. We get the Field objects of the two text fields in lines (1) and (2), and get the value (the `Field.value`) of the input field in line (3). We check to see if the value is the empty string or not, then in line (8), we set the rich value of the output field to the rich text value of the input

¹A friend reports that this field does not work for Acrobat 7, oh well.

field □

As a final example of this section, we ask the reader, that's you, to enter a two word phrase in the text field, the JavaScript code will then format it as rich text, and output the formatted string to a rich text field.

Example 3.5. Get text field the user using a response dialog box, and populate this value in a text field.

Again, this example may not work for version 8.x of Acrobat and Adobe Reader.

The verbatim listing of the push button action follows.

```

1  var rtn = app.response({
2      cTitle: "AcroTeX PDF Blog",
3      cQuestion: "Enter a two word string, and don't fool me!"
4  });
5  if ( rtn != null ) {
6      var stripRtn = rtn.replace(/\s*/g,"");
7      if ( stripRtn != "" ) {
8          var aWords = rtn.split(" ");
9          if ( aWords.length != 2 ) {
10             app.alert("You're trying to fool me! I'm watching!"
11                 + " I said two words!\r\r"
12                 + "Using the default string.");
13             aWords = [ "AcroTeX", "Rocks!" ];
14         }
15         var spans = new Array();
16         spans[0] = new Object();
17         spans[0].alignment="center";
18         spans[0].text = aWords[0]+" ";
19         spans[0].fontStyle = "italic";
20         spans[0].textColor = color.red;
21         // another method of populating the span elements
22         spans[1] = { text: aWords[1], fontStyle: "normal",
23             fontWeight: 700, textColor: ["RGB", 0, .6, 0 ],
24             underline: true }
25         var f = this.getField("myTxt3-5");
26         f.richValue = spans;
27     }
28 }

```

Comments. Once we get the user's response, we split the response into an array (8); if the

length of the array `aWords` is 2, we continue, otherwise, we put up an alert, and use a default message (13). To work with rich text, we need to build a span object, and that is what we do in lines (15)–(24). We create spans using two methods, the first method is demonstrated in lines (16)–(20), and in lines (22)–(23) we use another method. Once the span has been built, we get the field object of the target field in line (26), and set the value for that field using the `Field.richValue` property in line (26).

See the Span object in *JavaScript for Acrobat API Reference* for documentation of the properties used above. □

4. Changing the Appearance and the Properties of a Text Field

While we're at it, we may as well survey the properties and methods of the Field object that change the appearance and properties of a text field. In the next three sections we examine the properties, as listed in the Text Field Properties dialog box; in particular, we'll survey the General, Appearance, and the Options tab. The relation between the user interface terminology and the JavaScript properties and methods. Some simple examples are presented.

4.1. The General Tab

Below is a screen shot of the General tab of the Text Field Properties, parallel to the dialog box, are the JavaScript properties that correspond to these text field properties.

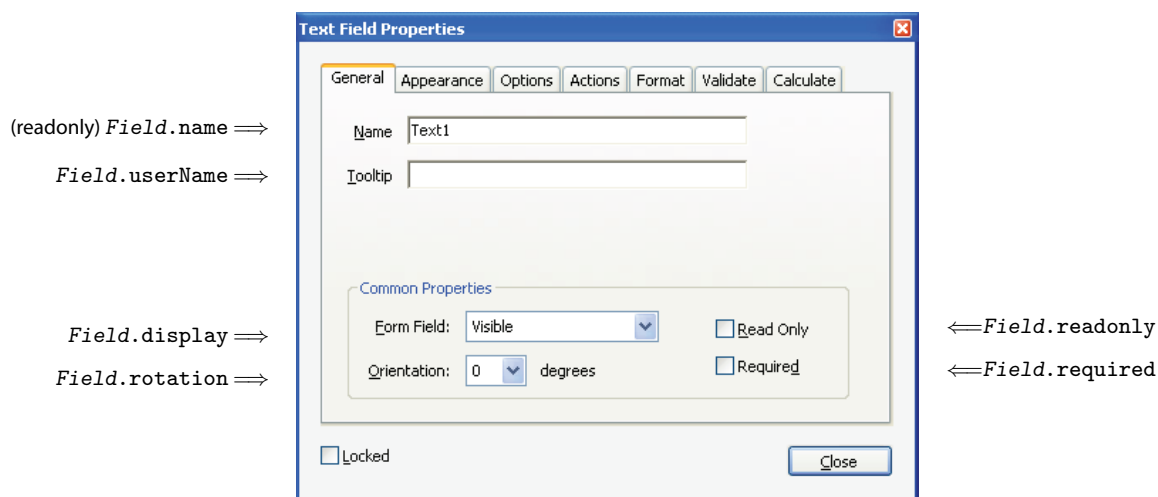


Figure 1: General Tab of the Text Field Properties

Use these properties by first acquiring the Field object of the target field, and assigning values

to the JavaScript properties. In general, some properties and methods are only available in Acrobat, and not available on Adobe Reader. See the *JavaScript for Acrobat API Reference* for full documentation, pay attention to the quick keys that accompany the properties and methods in the *JavaScript for Acrobat API Reference*.

Example 4.6. Toggle the readonly property of a text field.

The verbatim listing of the push button action follows.

```

1 var f = this.getField("myTxt4-6");
2 f.readonly = !f.readonly;
3 f.value = (f.readonly) ? "Field is readonly now"
4       : "You can enter data in the field if you wish";

```

Comments. After acquiring the Field object, we toggle the readonly property at line (2), we then send a message to the target field.

This example should work should work in Adobe Reader, but any changes to the document will not be preserved. □

4.2. The Appearance Tab

Let's do the same now for the appearance tab, which appears below.

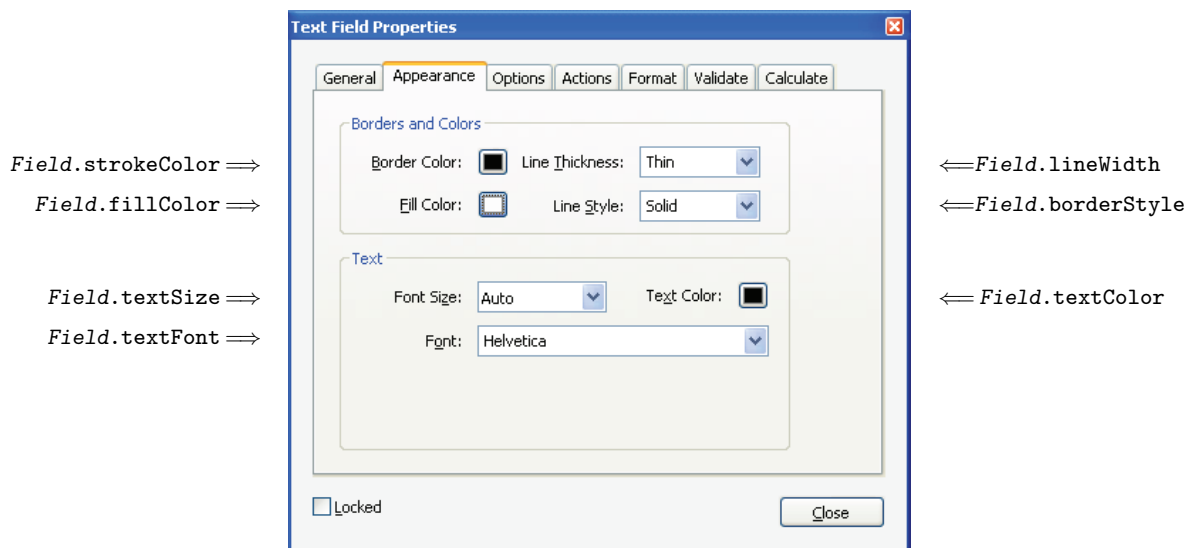


Figure 2: Appearance Tab of the Text Field Properties

As with the General tab, use these properties by first acquiring the Field object of the target field, and assigning values to the JavaScript properties. In general, some properties and methods are only available in Acrobat, and not available on Adobe Reader. See the *JavaScript for Acrobat API Reference* for full documentation, pay attention to the quick keys that accompany the properties and methods in the *JavaScript for Acrobat API Reference*.

Example 4.7. Cycle through some border colors, some border styles, and some text sizes.

The verbatim listing of the push button action follows.

```
1 var f = this.getField("myTxt4-7");
2 f.delay=true;
3     f.borderStyle= aFieldValues[nIndex][0];
4     f.borderColor=aFieldValues[nIndex][1];
5     f.textColor=aFieldValues[nIndex][1];
6     f.lineWidth=aFieldValues[nIndex][2];
7 f.delay=false;
8 nIndex = ++nIndex % aFieldValues.length;
```

Comments. There is a document level array, `aFieldValues`, and an index variable `nIndex`, that this script uses. These definitions are

```
var nIndex = 0;
var aFieldValues = new Array(
    [ border.s, color.red, 1 ],
    [ border.b, color.blue, 2 ], [ border.d, color.green, 3 ],
    [ border.i, color.magenta, 4 ], [ border.u, color.cyan, 1 ]
);
```

We get the Field object in line (1), in lines (2)–(7) we change the various appearance properties using the values on the `aFieldValues` array. Note that we used `f.delay=true` in line (2) and followed up with `f.delay=false` in line (7); this delays the re-drawing of the field until all the changes are set, we allow the changes to go through in line (7). See *Field.delay* in the *JavaScript for Acrobat API Reference* for details. Finally, we increment the index variable `nIndex` and perform mod `aFieldValues.length` arithmetic; this will either increment the variable, or if it increments to `aFieldValues.length`, it will set `nIndex` back to zero, so you, my beloved reader, can continue to click the button as much as you please.

The array `aFieldValues` and index variable `nIndex` so they are scanned once when the document is opened in the viewer. Had `nIndex` been put in as Field script, it would have been reset to zero each time the push button is pressed. □

I grow weary, but I now see the end of this digital torment; I'll keep marching on...to the Options tab.

4.3. The Options Tab

We set up the correspondence between each user interface element and its JavaScript counterpart.

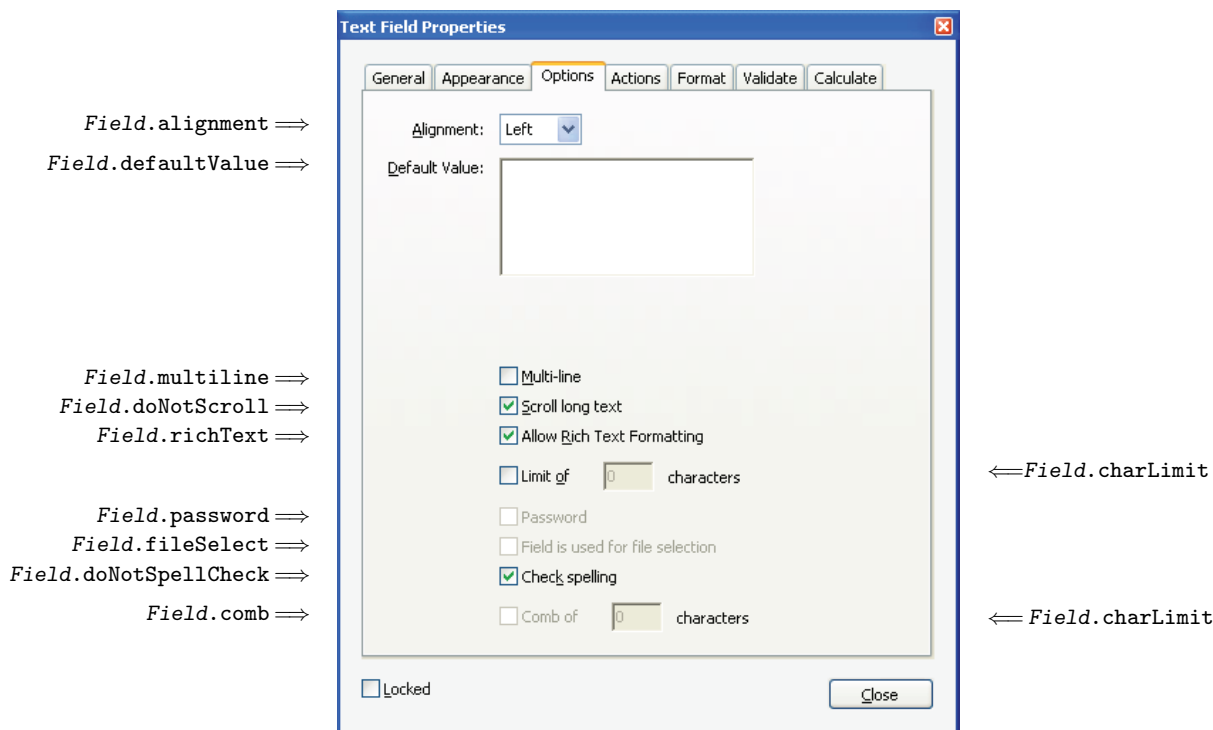


Figure 3: Options Tab of the Text Field Properties

Alignment: Controls how the text is laid out, possible values of `Field.alignment` are "left", "center", and "right".

Default value: See 'Setting the Default Value of a Text Field' on page 11.

Multiline: If this box is checked, the text field accepts multiline input. `Field.multiline` takes values of true and false, if true, the field accepts multiline input.

Scroll long text: To scroll or not to scroll, that is the question. The default is to scroll. Set `Field.doNotScroll` to false if you do not want the field to scroll.

Allow Rich Text Formatting: Check or clear this box using `Field.richText`, a property that

takes values of `true` and `false`. `Field.richText` is also used to determine if a text field accepts rich text formatting.

Password: If checked, the input appears as a series of `*` characters. `Field.password` takes a Boolean value to get or set this property.

Field is used for file selection: Setting `Field.fileSelect` to `true`, allows the user to enter a file path into the text field to be submitted. The user can also browse for the file if a push button is supplied that executes `Field.browseForFileToSubmit`. When `Field.browseForFileToSubmit` is executed, the user browses for a file, the path of the selected file is entered into the file selection field.

Check spelling: The default is to allow spell checking of the text input field. To disable spell checking, set `Field.doNotSpellCheck` to `false`.

Check Comb of characters: If `Field.comb` is set to `true`, a comb field is created. See [Example 4.9](#), page 11.

Limit of characters: Set `Field.charLimit` to a positive integer value to limit the number of characters input into the field. The property also sets the number of comb fields, when `Field.comb` is set to `true`.

Once again, check in with the [JavaScript for Acrobat API Reference](#) for full documentation on each of these properties.

• Creating a Comb Field

Example 4.8. Create a comb field out of an ordinary text field. Toggle between the two states.

The verbatim listing of the push button action follows.

```
1 var f = this.getField("myTxt4-8");
2 if ( f.comb ) {
3     f.comb=false;
4     f.charLimit=100;
5 } else {
6     f.comb=true;
7     f.charLimit=16;
8 }
```

Comments. If the field is a comb field, we put `f.comb=false` and put the `f.charLimit` to a large value, in lines (3)–(4); otherwise, we put `f.comb=true` and set the `f.charLimit` to 16.

Try as I might, I cannot make the “Limit of” check box in the Options tab deselect. I tried

putting `f.charLimit=0`, but this throws an exception. It may be a bug, or maybe, the Acrobat developers never thought that the famous [AcroTeX PDF Blog](#) would be toggling between a limit on the number of characters and no limit. There is no JavaScript property to check or uncheck that box. Putting `Field.charLimit` to a positive value checks the box, but I haven't found a way of unchecking. We'll settle with putting `f.charLimit=100`. □

• Setting the Default Value of a Text Field

The default value of a text field is the value the field displays when it is reset, perhaps using the `Doc.resetForm` method. For a text field, use `Field.defaultValue` to set this default value. The user interface for the default value is found in the Options tab, see [Figure 3](#) on page 9.

The default value for a rich text field also uses `Field.defaultValue` in conjunction with `Field.defaultStyle`.

Example 4.9. Toggle between setting a default value, and having no default value.

The verbatim listing of the push button action follows.

```

1 var f = this.getField("myTxt4-9");
2 var isDVClear = (f.defaultValue == "");
3 f.defaultValue = (isDVClear) ? "AcroTeX PDF Blog" : "";
4 var msgStr="The default value "
5     + ((isDVClear) ? "is \"AcroTeX PDF Blog\"." : "has been cleared.")
6     + "\r\r Now press the Reset button to see the new default value."
7 app.alert(msgStr);

```

You'll note that the default value of the field is not necessarily the value of the field. Even after the default value is set, it does not appear in the text field for the first time until the field is reset. Certainly, when the document is opened, the above text field has no value. To have a value visible in the field, you need to give it a value, using `Field.value`, the user interface, or, as I do, a `pdfmark` operator from within the authoring application, which in my case is \TeX . □

These exercises of setting properties using JavaScript become considerably more interesting when you dynamically create a field using the `Doc.addField` method, but that will have to wait for another blog.

That's pretty much it for now, I simply must get back to my retirement. $\text{\textcircled{D}}$