



AcroTeX.Net

AcroTeX PDF Blog

Processing Acrobat Forms using JavaScript

External Processing of a Field

Part 3: Button Fields, the Push button

D. P. Story

5. Button Fields: The Push button

A button field is an interactive control the user manipulates with the mouse. Button fields do not take keyboard input from the user. There are three types of buttons fields,¹

1. A *Push button* is a purely interactive control that has not value.
2. A *check box* toggles between two states, on and off.
3. *Radio button fields* contain a set of related buttons that can each be on or off. Normally, when one radio button is on, the others in the same field are off.

In this article, we shall concentrate all our efforts on the push button, leaving the others for subsequent blog articles.

5.1. Getting and Setting the Value of a Push button

The push button has no value, it is neither on or off so you cannot get or set the value of a push button.

5.2. Changing the Appearance and the Properties of the Button Field

Until such time arrives that we begin to examine the Actions tab—where JavaScript actions are defined—we shall concentrate on the appearance and properties of a button field.

• The General Tab

Below is a screen shot of the General tab of the Button Properties, parallel to the dialog box, are the JavaScript properties that correspond to the elements on the General tab of the Button Properties dialog box.

The properties of the General tab, [Figure 1](#) on page 3, are the same as those of the other fields. We'll present an example that illustrate properties different from the ones given for the text field ([AcroTeX Blog #13](#)).

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

¹The descriptions of button fields are paraphrased from Section 8.6.3 of the *PDF Reference, Sixth Edition, version 1.7*.

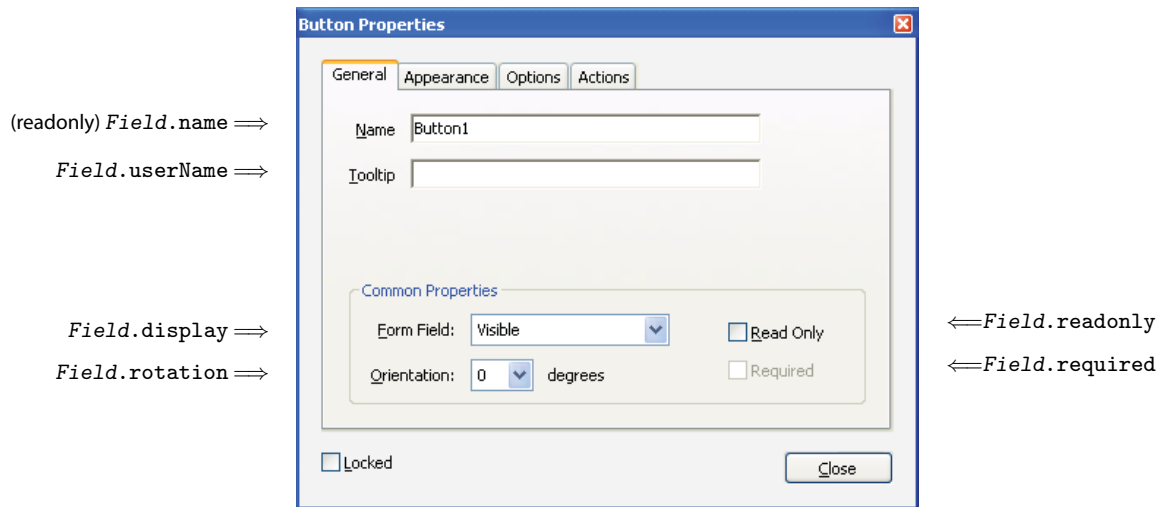


Figure 1: The General Tab of Button Properties

documentation, pay attention to the quick keys that accompany the properties and methods in the *JavaScript for Acrobat API Reference*.

Example 5.1. Use the button on the left to toggle the button on the right between visible and hidden. This uses the *Field.display* property.

The verbatim listing of the push button action follows.

```

1 var f = this.getField("myPBRight5-1");
2 var g = this.getField("myTxt5-1");
3 if ( f.display == display.visible ) {
4     f.display = display.hidden;      // make button hidden
5     g.value = "Now you don't!";     // send out a message
6 } else {
7     f.display = display.visible;    // make button visible
8     g.value = "Now you see it!";    // send out a message
9 }

```

□

When a push button is set to readonly (*Field.readonly=true*), the button is no longer interactive. Why would you want a button not to be interactive? Well, a button can have an icon as an appearance, and often times this is a cheap way of creating an image. Examples appear later.

• The Appearance Tab

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

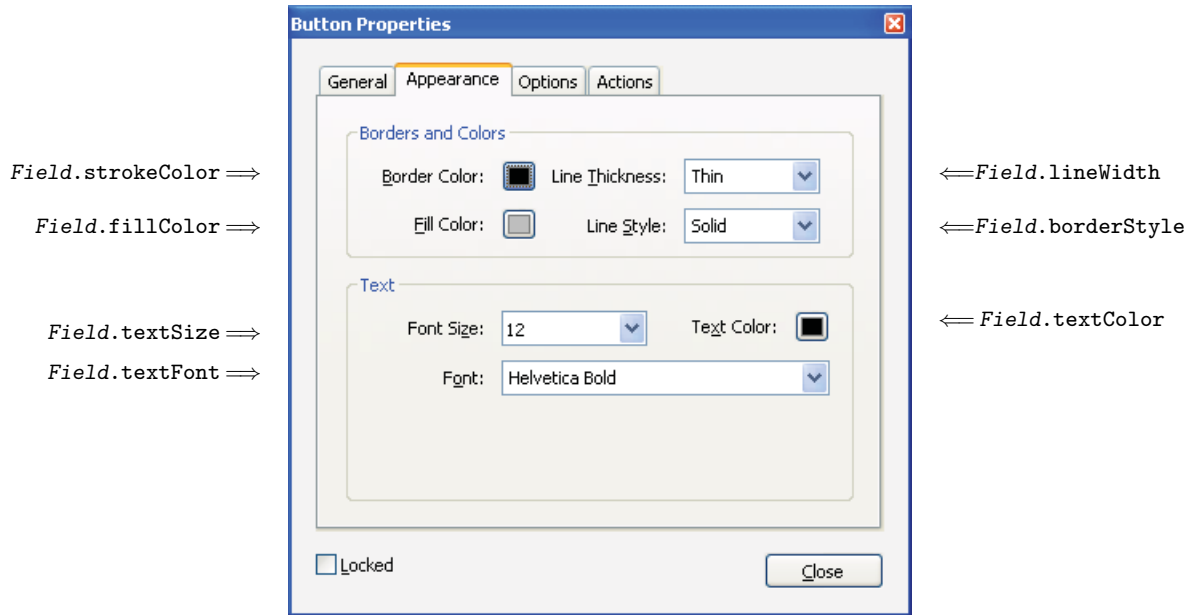


Figure 2: The Appearance Tab of Button Properties

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 5.2. Choose a color for the text on the button.

The verbatim listing of the push button action follows.

```

1 var c = this.getField("myCombo5-2");
2 var f = this.getField("myPB5-2");
3 f.textColor=eval(c.value);

```

Comments. We get the Field objects in line (1) & (2) of the combo box and the right-most push button; in line (3), we set the `f.textColor` property of the push button equal to `eval(c.value)`.

A digression (push the return address). A combo box has two values, the *export value* and

the *face* (or *appearance*) value; in this case, `c.value` is the export value (this is the value you don't see). For example, if you select Blue from the combo box, the word "Blue", a string, is the face value, while the export value is `"color.blue"`, is also a string. Press the Select Object tool on the tool bar, and double-click on the combo box to view its properties, in particular, view the Option tab where the items in the combo box are list along with their export values.

Pop the return address. To understand line (3) better, suppose you have chosen the color Blue from the combo box. When we evaluate the export value `eval(c.value)`, we are evaluating `eval("color.blue")`, which evaluates to `color.blue`. This is the blue property of the color object, described in *JavaScript for Acrobat API Reference*, in the section on the Color object. The bottom line, `eval(c.value)` evaluates to a color that `Field.textColor` likes and can use. □

• The Options Tab

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

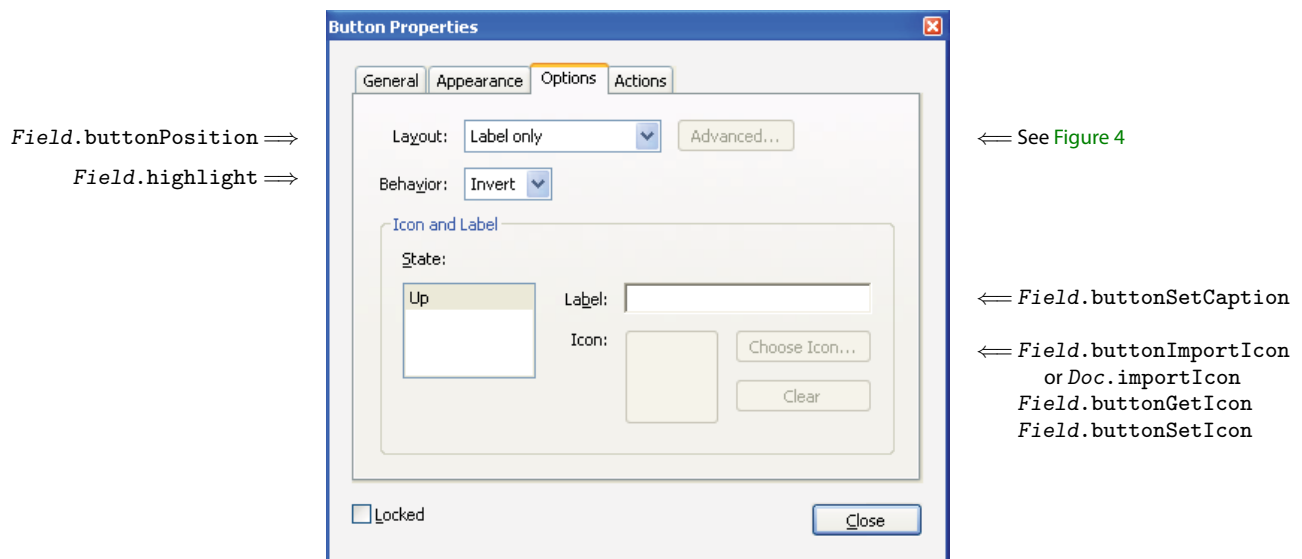


Figure 3: The Options Tab of Button Properties

Layout: `Field.buttonPosition` controls how the text and icon are positioned relative to each other, possible values are `position.textOnly` (Label only), `position.iconOnly` (Icon only), `position.iconTextV` (Icon top, label bottom), and `position.textIconV` (Label top, icon bottom), `position.iconTextH` (Icon left, label right), `position.textIconH` (Label left, icon right), and `position.overlay` (Label over icon).

Behavior: *Field.highlight* determines how a button reacts when the user clicks it. Possible values are *highlight.n* (none), *highlight.i* (invert), *highlight.p* (push), and *highlight.o* (outline).

The *Field.highlight* is set to none, invert, or outline, there is only one state, Up; for the push behavior there three states Up, Down, and Rollover.

Advanced...: When the *Field.buttonPosition* is set to a value with an icon, additional Advanced properties are available. See [Figure 4](#) for details.

Label: The label that appears on the button is set by *Field.buttonSetCaption*, this is a Field method, one argument sets the label, the other argument sets the state the label applies to.

Choose Icon: When the user selects a Layout that involves an icon, the Advance button and the Choose Icon button are enabled; recall that JavaScript selects the Layout through the *Field.buttonPosition*, when the value includes an icon specification, one or more icons needs to be specified. This is normally done through Select Icon dialog box of the user interface. If we intend to go totally native, we need to use *Field.buttonImportIcon* or *Doc.importIcon* to import an icon; *Field.buttonGetIcon* to get the icon object, and *Field.buttonSetIcon* to set the icon in the desired appearance state (Up, Down, or Rollover). Examples are found below.

Example 5.3. This example uses *Field.buttonImportIcon* to import an icon for a button face. This was the first of the import icon-type methods, and is not as good to use as *Doc.importIcon*, as we shall see.

The verbatim listing of the push button action appears to the right.

```
1 var f = this.getField("myPBIcon5-3");
2 f.buttonPosition=position.iconOnly
3 f.buttonImportIcon();
```

This example requires Acrobat. Use the button to import your own icon.

Comments. We get the Field object, as usual, then give the push button a icon only property, using *Field.iconOnly*. We then execute *f.buttonImportIcon()*; , this button has no restrictions on it as long as argument of the method is empty. The use of this method requires Acrobat, not Adobe Reader.

This method will import the icon the user selects and put it as the normal or Up appearance. If the Behavior of the button is Push (meaning there are three states Up (normal), Down, and

Rollover), the icon will appear in all three states. If there are also captions for each state; however, the icon will only appear in the normal state, as the next example shows. □

Example 5.4. This example uses *Field.buttonImportIcon* to import an icon for a button face, *Field.buttonPosition* to set the button for a Layout of icon on top and label below, and *Field.buttonSetCaption* to set the labels.

The verbatim listing of the push button action appears to the right.

```

1  var f = this.getField("myPBIcon5-4");
2  f.buttonPosition=position.iconTextV
3  f.buttonImportIcon();
4  f.buttonSetCaption("DPS",0);
5  f.buttonSetCaption("AcroTeX Rocks!",1);
6  f.buttonSetCaption("says",2);

```

This example requires Acrobat. Use the button to import your own icon.

Comments. The appearance is always for the normal or Up appearance face, when you use *Field.buttonImportIcon*. Since there is an appearance for the Down and Rollover states (the labels), the icon we imported does not appear. □

To get an icon appearance for all three states using *Field.buttonImportIcon*, we need to use *Field.buttonGetIcon* and *Field.buttonSetIcon*, ...and some trickery.

Example 5.5. This example uses *Field.buttonImportIcon* to import an icon for a button face, *Field.buttonPosition* to set the button for a Layout of icon on top and label below, and *Field.buttonSetCaption* to set the labels. We use a hidden push button to import the icons, then transfer them to our target button. Wish me luck, here we go!

The verbatim listing of the push button action appears to the right.

```

1  var f = this.getField("myPBIcon5-5");
2  var g = this.getField("myPBHidden5-5");
3  g.buttonPosition=position.iconOnly;
4  f.buttonPosition=position.iconTextV;
5  g.buttonImportIcon();
6  var oIcon=g.buttonGetIcon(0);f.buttonSetIcon(oIcon,0);
7  g.buttonImportIcon();
8  oIcon=g.buttonGetIcon(0);f.buttonSetIcon(oIcon,1);
9  g.buttonImportIcon();
10 oIcon=g.buttonGetIcon(0); f.buttonSetIcon(oIcon,2);
11 f.buttonSetCaption("DPS",0);
12 f.buttonSetCaption("AcroTeX!",1);
13 f.buttonSetCaption("is",2);

```

If you have Acrobat, you can try this button and import your own favorite images.

Comments. Basically, the workflow is to get an icon from the user, and put it in the hidden

field. Then use `Field.buttonGetIcon`, lines (6), (8), and (10) to get the icon object of the newly imported icon and to set that icon object as the appearance face of our target button using `Field.buttonSetIcon` □

Using `Doc.importIcon`: You can see from the previous you have to be pretty sneaky to get the icons into the document for several button faces. The `Doc.importIcon` is a much better way of importing icons into the document, it allows you to name the icons so that can be used as button faces for may different buttons, without increasing the file size of your PDF. Here is one final example on this topic.

Example 5.6. Import icons using `Doc.importIcon` and `Doc.getIcon` along with the method `Field.buttonSetIcon`.

```

1  var f = this.getField("myPBIcon5-6");
2  f.buttonPosition=position.iconTextV
3  f.delay=true;
4      this.importIcon("normal");
5      var oIcon = this.getIcon("normal");
6      f.buttonSetCaption("Normal",0);
7      f.buttonSetIcon(oIcon,0);
8      this.importIcon("down");
9      f.buttonSetCaption("Down!",1);
10     oIcon = this.getIcon("down");
11     f.buttonSetIcon(oIcon,1);
12     this.importIcon("rollover");
13     f.buttonSetCaption("Rollover",2);
14     oIcon = this.getIcon("rollover");
15     f.buttonSetIcon(oIcon,2);
16  f.delay=false;

```

If you have Acrobat, you can try this button and import your own favorite images.

Comments. We import named icons into the document in lines (5), (8), and (12), naming them appropriately. For each of these icons, we get them by name in lines (6), (8), and (14). Then, we set them as icon appearances in lines (7), (11), and (15). As we do all these things, we also set the captions for each of the appearance states.

To illustrate the destructive nature of `Field.buttonImportIcon()`, if we were to execute `f.buttonImportIcon()` just after line (15), that would destroy the three appearances we meticulously created, and we are left with a single appearance, the one imported by `f.buttonImportIcon()`.

One the icons are imported as named icons, they can be used elsewhere in the document. □

Enough of that for now. There is another user interface element what I've yet to mention. Back in [Figure 3](#), page 5, where was the Advanced button that is enabled when icons are used.

Figure 4 shows the Icon Placement dialog box, along with the JavaScript counterparts.

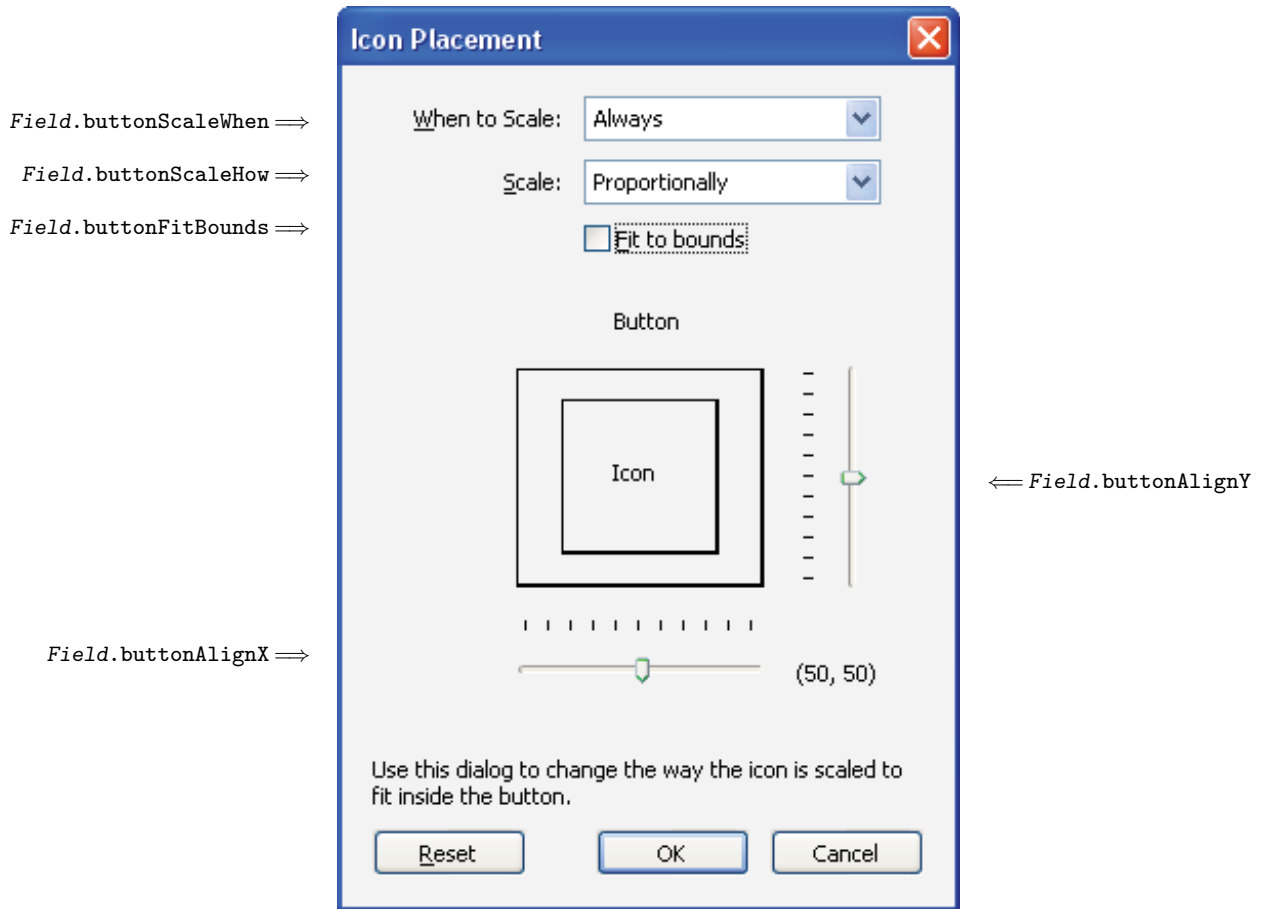


Figure 4: Icon Placement Dialog Box

I won't say much about these, this blog has gotten too long. We can use the referenced JavaScript properties to position the icon within the bounding rectangle.

5.3. An Exciting Example

AcroSort is a \LaTeX package of mine for creating a novelty effect. When this document is built, “disposable” code is used then discarded. This script uses `Doc.importIcon` to imported a tiled version of a photo, then uses `Doc.addField` to create a grid of push buttons, then associates with each field and one of the icon tiles. When you press the start button, the tiles are randomly mixed, then a bubble sort is used to properly order them in the frame. Very sweet.

The tiling of the photo is done using a batch sequence I wrote for this purpose. Originally, I tried tiling using Adobe Illustrator, this worked, but I am out of my element iwth AI. Instead, I wrote a batch to tile a PDF page. The batch sequence, called the AeB Slicing batch sequence, is freely available at the AeB Pro web site.² These you will find other goodies that may be of interest to you. Check out the AcroMemory demo files as well. This too is based on importing icons for appearances, then using them to create a game.

My Acro \TeX web site has some other offerings in games that use Acrobat forms.³,
Well, that’s pretty much it for now, I simply must get back to my retirement. $\text{\textcircled{S}}$

²http://www.math.uakron.edu/~dpstory/aeb_pro.html

³<http://www.math.uakron.edu/~dpstory/acrotex.html>