AcroTEX.Net

AcroTEX PDF Blog

Processing Acrobat Forms using JavaScript

Internal Processing of a Field

Part 1: The Event Object and the Action Tab

D. P. Story

Copyright © 2008 dpstory@acrotex.net
AcroTEX PDF Blog #20

http://www.acrotex.net
Published: December 31, 2008

Table of Contents

12 The Event Object 3

12.1 The `type` Property of the Event Object 3

12.2 The `name` Property of the Event Object 4

12.3 Actions associated with Triggers 5

12.4 Properties of the Event Object 7

13 The Action Tab 8

13.1 Using `event.target` 8

13.2 Using `event.value` 11

- Using `event.value` with Mouse Events 11
- Using `event.value` with Focus and Blur Events 12

13.3 AcroTEX Peg Solitaire 14

www.AcroTeX.net

12. The Event Object

It is written in the section on the `event` object in the *JavaScript for Acrobat API Reference* that "all JavaScript are executed as a result of some *event*." An event is usually instigated by user interaction with the document. When an event occurs, the Acrobat JavaScript engine creates an `event` *object* that has properties appropriate to that event. The `event` object can have many properties, but we mention only two at this time: the `type` and `name` properties. These two properties are used by Acrobat documentation to classify, or characterize, the event.

The *JavaScript for Acrobat API Reference* classifies all events by the `type/name` combination. The reader is referred to the section titled Event type/name combinations, which provides detailed descriptions of all events (`type/name` pairs), and lists the properties of the `event` object created.

The `event` object enables the script developer to write general scripts that can be used and re-used in one or more classes of form fields. The event object itself contains information about the field, there is not a strong need to include in the script anything that is specific to a particular field.

A more detailed discussion of `type` and `name` follows in sections 12.1 and 12.2.

Internal Processing. What does the `event` object have to do with the current blog topic on *internal processing of fields*? When a field wants to investigate itself—get or set its own current value, get or set own appearance, and so on—, it uses the `event` object. This is, in my mind, the primary distinction between external and internal processing:[1]

- if a script *does not use* the `event` object, it is *external*;
- if a script does use the `event` object, it is *internal*.

Of course, a script can use the event object (internal processing), and process other fields as well (external processing). Internal and external are convenient ways of organizing categories of scripts.

12.1. The `type` Property of the Event Object

Events are organized by *type* (or *category*, which might be a more descriptive term). The event type describes what construct of Acrobat the user in interacting with. The following are the different types (or categories) of events, these are the ones that appear as the string value of the `event.type` property:

[1] External and internal processing are terms I coined for this blog, I haven't seen them used elsewhere.

AcroTeX PDF Blog

- `App`: An event that occurs when the PDF viewer (Adobe Reader, or some flavor of Acrobat) is started.
- `Batch`: An event initiated by the user starting a batch sequence, a feature of Acrobat Pro (Extended).
- `Bookmark`: An event initiated by the user clicking on a bookmark.
- `Console`: An event that occurs whenever a user evaluates JavaScript in the JavaScript Debugger Console window, a feature of Acrobat.
- `Doc`: An event that occurs the document is opened, closed, saved, printed, etc.
- `External`: An event that is the result of external access to the document, perhaps, through OLE, AppleScript, or loading of an FDF (Forms Data Format).
- `Field`: An event that occurs as a result of user interaction with a Acrobat form field. There is a number of ways that may happen, by bringing focus to the field, by losing focus (blurring the field), clicking the field, entering data into the field, and several others that are detailed in this and in later articles.
- `Link`: An event that occurs when a user clicks a link.
- `Menu`: An event that occurs whenever JavaScript that has been attached to a menu item is executed.
- `Page`: An event that occurs when a page is opened or closed.
- `Screen`: An event that occurs when the user interacts in various ways with the legacy multimedia screen annotation.

If the user interacts with a field, we say that a *Field event* occurs, if a page is changed, a *Page event* occurs, and so on.

This blog thread concerns mostly Acrobat form fields; consequently, we'll focus on Field events.

12.2. The `name` Property of the Event Object

Each type of event may be *triggered* in one or more ways. Each trigger has a *name*; the trigger name is the string value of the `event.name` property. The trigger name is the name of the action (usually by the user) that initiated the event. For Field type events, the trigger names are

```
1 Focus, Blur
2 Mouse Down, Mouse Enter, Mouse Exit, Mouse Up
3 Keystroke
4 Format, Validate, Calculate
```

Triggers listed in lines (1)–(2) are common to all fields. The `Keystroke` trigger, line (3), is common to text fields, combo boxes, and list boxes, while the triggers listed in line (4) occur in text fields and editable combo boxes.

www.AcroTeX.net

AcroTeX PDF Blog

www.AcroTeX.net

To illustrate the terminology,

- A user clicks a push button, a Field event object is created, the name of the triggering action is `Mouse Up`.
- A user clicks a link, a Link `event` object is created, the name of the triggering action is still `Mouse Up`. The `event` object for the Field event and the `event` object for the Link event share some properties, but not others.
- A user enters presses the letter "A" while a text field is in focus, a `Keystroke` triggers the creation of a Field `event` object.
- A certain field is in focus, and the user presses the tab key, this triggers a Blur of the field that is in focus.
- The user enters a number into a text field and presses the Enter key to indicate the input is complete, a `Validate` trigger is activated to check whether the number entered is within the required range of values.

12.3. Actions associated with Triggers

The whole point of all this is to associate an *action* with a *trigger* so that desired tasks can be performed. When the user interacts with a field by clicking on it, for example, there should be an action that occurs as a result. There can be non-JavaScript actions, but these are of no interest to us JavaScript fanatics. Yes, JavaScript is the type of action we crave.

To associate a JavaScript action with a particular field (event) and trigger, bring up the properties dialog box of the field and select the Action tab, see Figure 1(a) on page 9. The triggers are listed under the Select Trigger option menu (see Figure 1(b)), there you will see Mouse Up, Mouse Down, Mouse Enter, Mouse Exit, On Focus, and On Blur. These correspond to the triggers mentioned earlier. To create a JavaScript action, select Run a JavaScript from the Select Action option menu (see Figure 1(a)). After the trigger and action type have been selected, click on the Add button, and the JavaScript editor appears for your coding pleasure.

The rest of this article, beginning with section 13 on page 8, is devoted to the Form type event triggered by `Mouse Up`, `Mouse Down`, `Mouse Enter`, `Mouse Exit`, `Focus`, and `Blur`. Why these? These are the triggers associated with the *Action tab* of the properties dialog box of an Acrobat form field. The Action tab is common to all the fields we've previously studied,[2] they are the simplest to understand and to work with.

The next example illustrates the `type` and `name` properties of the `event` object. Each trigger, including Link event for opening the console, has an associated JavaScript action, each executes the `printEventTypeName()` function defined as document JavaScript as follows:

[2]The signature field, which we have and will not study, has an Action tab as well.

AcroTeX PDF Blog

```
function printEventTypeName()
{
    console.println("event.type = " + event.type);
    console.println("event.name = " + event.name);
    console.println("--------------------------");
}
```

Example 12.1. Manipulate the button below. There are JavaScript actions defined for the triggers `Mouse Down`, `Mouse Enter`, `Mouse Exit`, `Mouse Up`, `Focus`, and `Blur`. Open the console window first:

You will note the following events occurs when you move your mouse over the button, click and release your mouse, exit the button's bounding rectangle, and click on a white space outside the bounding rectangle.

```
1  event.type = Field            Enter bounding rectangle
2  event.name = Mouse Enter
3  --------------------------
4  event.type = Field            Left mouse button depressed
5  event.name = Mouse Down
6  --------------------------
7  event.type = Field            Field receives focus
8  event.name = Focus
9  --------------------------
10 event.type = Field            Release left mouse button
11 event.name = Mouse Up
12 --------------------------
13 event.type = Field            Exit bounding box of button
14 event.name = Mouse Exit
15 --------------------------
16 event.type = Field            Click on what space, field loses focus as a result
17 event.name = Blur
18 --------------------------
```

If we exit the bounding rectangle with the left mouse button still depressed (without ever releasing it) the Mouse Up trigger never fires, so line (10)–(11) do not occur. If you enter and exit the bounding rectangle of the button without pressing the left-mouse button, the field never gets focus, assuming it wasn't already in focus. The field can be brought to focus without the entering with the mouse and left-clicking; you can gain focus by using the tab key and tabbing into the field. It will lose focus (blur) when you tab away from the field (or by clicking else where external to the bounding box of the button). □

Working with the `event` object, its properties takes a good deal of understanding, experience, and practice. We'll detail the Field events with its various triggers in this thread of blogs on

www.AcroTeX.net
AcroTeX PDF Blog

www.AcroTeX.net

internal processing of fields.

12.4. Properties of the Event Object

The properties of the event object vary depending on the particular `type/name` combination. The `type` and `name` properties are most certainly common to all events. In this section we list other properties that are common to all events. These are used in some of the examples that follow in section 13.

- `target`: The `event.target` property, for Field events, is the Field object that triggered the event. For events other than Field events, `target` is the Doc object (`this` object).

  Through the `event.target` property, we gain access to the Field object of the field, and, as a consequence, all its properties and methods.
- `targetName`: The `event.targetName` property, for Field events, is the name of the field that triggered the event. For Field events, `event.targetName=event.target.name`. For other events, `targetName` has a different meaning.
- `shift`: The `event.shift` property returns `true` if the shift key is down during the event, and returns `false`, otherwise.

  Allows the script writer to offer an alternate action, if the shift key is depressed.
- `modifier`: The `event.modifier` property returns `true` if the modifier key is down during the event, and returns `false`, otherwise. The modifier key for a Windows platform is Control, and for Mac OS is Option or Command.

  Allows the script writer to offer an alternate action, if the modifier key is depressed.

In addition to these, we mention two others (not available for all fields) that are extremely valuable in internally processing a field. These are

- `rc`: To quote *JavaScript for Acrobat API Reference*, `event.rc` is "used for validation. Indicates whether a particular event in the event chain should succeed. Set to `false` to prevent a change from occurring or a value from committing. The default is `true`."

  Not every event sets or listens to the `rc` property, section title Event type/name combinations documents the role the `rc` plays with each event type.

  In this particular blog article, which deals with the Action tab, we shall not be using the `rc` property because the various triggers for the Action tab do not listen to or set the `rc` property.
- `value`: The `event.value` property is of great value. `:-{)` The event.value property represents a "value" of the field as it is being edited by the user. Its meaning changes depending on the trigger (Validate, Calculate,Format, Keystroke, Blur, Focus). Some of

AcroTeX PDF Blog

www.AcroTeX.net

the examples in section 13 will illustrate this property for the Blur and Focus triggers. The other triggers are used through the Format, Validate and Calculate tabs of text fields and editable combo boxes, these will be dealt with in a later blog.

There is a whole plethora of other event properties that will be taken up all in good time and all in good AcroTEX PDF Blogs.

We finish this section with a simple example illustrating `event.shift` and `event.modifier`.

Example 12.2. **Riddle me this:** Who has three legs in the morning, two legs in the afternoon, and four legs in the evening?

That started out as a simple example, but blossomed into much more than I planned. The Ans button is the one that illustrates the `event.shift` and `event.modifier` properties. I leave it to you to view the script attached to the Ans button. (To show the verbatim listing would reveal the answer.)

You might look at the `Keystroke` event associated with text field (the input field for your response). I may use this example in a later blog to illustrate the `Keystroke` trigger. □

13. The Action Tab

The Action tab, Figure 1, is the work horse of the tabs for attached JavaScript, the others being `Format`, `Calculate`, and `Validate`. While the Action tab is primarily designed for external processing (no use of the `event` object, see 'AcroTEX Peg Solitaire' on page 14 for an example), it can do internal processing as well, as will be illustrated in this section. The other three tabs (`Format`, `Calculate`, and `Validate`) are designed for internal processing, these will be taken up in later blogs.

To attach a JavaScript to a Field event with a particular trigger, you must set the Select Action option list to Run a JavaScript, as shown in Figure 1(a). The trigger is set with the Select Trigger option list, see Figure 1(a), there you see the usual triggers discussed earlier.

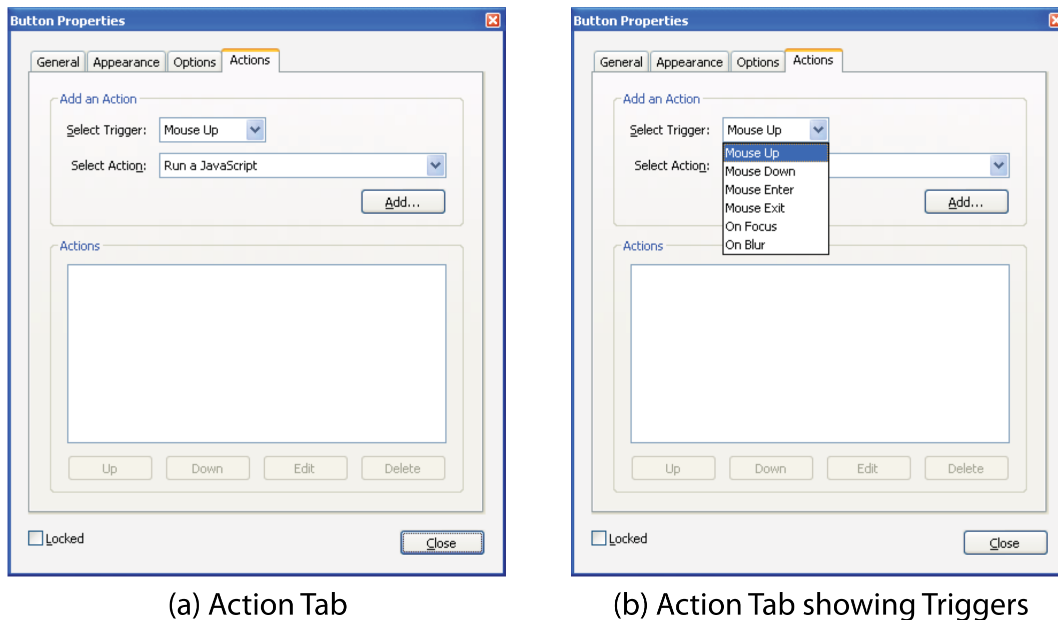
Having chosen your trigger, click the Add button, and the JavaScript editor will appear. After you've finished entering your code, click OK, then OK again, to back out of the editor and dialog box.

13.1. Using `event.target`

The `event.target` property brings back the Field object that is running the script.

Example 13.3. Changing the appearance of a button and text field with `event.target` and the `Mouse Up`, `Mouse Down`, `Mouse Enter`, and `Mouse Exit` triggers.

AcroTeX PDF Blog



(a) Action Tab (b) Action Tab showing Triggers

Figure 1: The Action Tab, common to all Acrobat Form Fields

All triggers for the two fields execute the same JS function `changeColorsTo(myColor)`, implemented as document JavaScript, and listed below. Though the same script is executed, the behavior of the button and text field differ:

- The `Mouse Enter` event (rollover) is the same for both fields; however, the rollover caption for the button does not appear. The rollover button caption is `"Me"`; I'm not sure what this is due to.
- The `Mouse Down` event is the same for both, the button caption is as it should be.
- The `Mouse Up` event is the same for both.
- For the `Mouse Exit` event, the button works as advertised, the original color is returned to the button, but for the text field, the border remains painted with the color set by the `Mouse Down` event, the color passed to it by the `Mouse Up` event never appears. It is only after the data is committed (click outside the field, or tab away from the field) that the `Mouse Exit` code takes effect. The reason for this can be more easily guessed: The appearance state of the text field is cached until the data is committed.

  To prove this last statement, enter some data in the text field, then press the `Enter` key, the text turns red, which is the color set by the `Mouse Up` event. When the data is committed, any changes in appearance is updated.

www.AcroTeX.net
AcroTeX PDF Blog

www.AcroTeX.net

The code that follows is quite general, it does not reference any particular field.

When the mouse enters the bounding rectangle, the current colors are saved as custom properties of the Field object, in this way, they will persist, and the original colors can be recovered on exit. On mouse exit, the original colors are returned to *Field*.strokeColor and to *Field*.textColor.

If the user tabs into the text (or button) field, a mouse event does not occur, and so no change in appearance. To counter this, I added a Focus and Blur event; Focus does the same as Mouse Enter, and Blur does the same as Mouse Exit.

```
1  function changeColorsTo(myColor) {
2      var f=event.target;
3      f.delay=true;
4      switch (event.name) {
5          case "Mouse Enter":
6          case "Focus":
7              // save current colors of this field
8              var isColorNotSaved=( f.thisStrokeColor==undefined );
9              if ( isColorNotSaved) {
10                 f.thisStrokeColor=f.strokeColor;
11                 f.thisTextColor=f.textColor;
12             }
13             f.strokeColor=f.textColor=myColor;
14             break;
15         case "Mouse Exit":
16         case "Blur":
17             // restore saved colors for this field
18             f.strokeColor=f.thisStrokeColor;
19             f.textColor=f.thisTextColor;
20             break
21         default:
22             f.strokeColor=f.textColor=myColor;
23     }
24     f.delay=false;
25 }
```

Comments: Get the Field object in line (2) using event.target. Set up a switch as a function of event.name. If the trigger (event.name) is either Mouse Enter or Focus (lines (5)–(13)), save the current colors and give them myColor, passed as an argument. The conditional statement in line (9) is needed because the Focus event occurs after Mouse Enter (assuming no tabbing); we only save the colors once, this prevents the Focus from saving them again and we end of with a color other than black as the saved color. If the trigger is Mouse Exit or Blur (lines (15)–(19)), restore the saved colors. Otherwise, line (19), change the colors to myColor.

AcroTeX PDF Blog

www.AcroTeX.net

The code can be used and re-used many times, you just pass what color you want to display in the appearance. One can generalize the function, of course, to include two colors, but I won't bother. □

Example 13.4. Toggle the caption of a button, for no apparent reason.

The code for this effect is short:

```
var f=event.target;
f.delay=true;
    var isAcroTeX=(f.buttonGetCaption()=="AcroTeX");
    f.textColor=(isAcroTeX)?color.red:color.blue;
    f.buttonSetCaption((isAcroTeX)?"Rocks":"AcroTeX");
f.delay=false;
```

Well, it started out simple. □

13.2. Using `event.value`

Now, we have to get away from (my beloved) buttons, because they have no value. We illustrate `event.value` on form fields that take on values, keeping in mind, we are restricing ourselves for now to the Action tab that is, to the triggers `Mouse Up`, `Mouse Down`, `Mouse Enter`, `Mouse Exit`, `Focus`, and `Blur`.

• Using `event.value` with Mouse Events

On close reading of `value`, we see there is no mention of mouse events, so `event.value` must be the (current) value of the field, right? Let's check it out.

Example 13.5. Exploring `event.value` for each form field type. Open the console window, use this link if you are using the AR.

- Text fields:
- Button Fields:
  - Push Button:
  - Check Box:
  - Radio Buttons:
- Choice Fields:
  - List Box: (multiselect)
  - Combo Box: (custom text allowed)

AcroTeX PDF Blog

www.AcroTeX.net

Is `event.value` the current value of the field? The answer is NO! It seems that in all the above cases, for the advertised triggers, `event.value` is the empty string! □

Important: For the triggers `Mouse Up`, `Mouse Down`, `Mouse Enter`, and `Mouse Exit`, event.value is the empty string, and thus, gives no information on the value of the field; however, the current value of the field can be obtained from `event.target.value`. The `event.target.value` property returns the export value, in the case of list boxes and combo boxes.

• Using `event.value` with Focus and Blur Events

For these two events, *JavaScript for Acrobat API Reference* has something to say under the description of `event.value`:

> For Field/Blur and Field/Focus events, it is the current value of the field. During these two events, `event.value` is read only. That is, the field value cannot be changed by setting event.value.
>
> Beginning with Acrobat 5.0, for a list box that allows multiple selections (see field.multipleSelection), the following behavior occurs. If the field value is an array (that is, multiple items are selected), event.value returns an empty string when getting, and does not accept setting.

Let's test it out on the same set of fields, shall we?

Example 13.6. Exploring `event.value` for each form field type. Open the console window, use this link if you are using the AR.

- Text fields:
- Button Fields:
  - Push Button:
  - Check Box:
  - Radio Buttons:
- Choice Fields:
  - List Box: (multiselect)
  - Combo Box: (custom text allowed)

Observation: The `event.value` property returns the value of the field (the export value, in the case of the list and combo boxes) at the time Focus was obtained, and gives the value (or export value) of the field at the time when focus is lost (Blur occurs).

AcroTeX PDF Blog

www.AcroTeX.net

To illustrate this, look at the checked check box. Use your mouse to enter the check box, *press and release* (Mouse Down and Mouse Up) the left mouse button, the check box is cleared, but the console window shows `event.value=AcroTeX`. Now exit the field by tabbing away, or clicking somewhere outside the field is blurred and the Blur event occurs, `event.value` returns a value of `Off`.

To understand this behavior, look at the flow chart in the section Form event processing. The Focus event occurs immediately *after* the Mouse Down event, and *before* the Mouse Up event. It is during the Mouse Up event that the check box changes its value. So on Focus, when the check box is checked, the value returned by `event.value` is the `On` value, which is `AcroTeX`. Only on Mouse Up, is the box cleared and the value of the field becomes `Off`. □

In my own work, I use Focus to initialize variables as the user enters a text field, for example, and on blur, cleaning up after the user has finished. Use these events in what ever way your imagination can conceive.

One the next page is a little application I did several years ago to amuse my son, Alexander. It is a game you find in some restaurants to while away the time waiting for your meal. See 'AcroTEX Peg Solitaire' on page 14.

AcroTeX PDF Blog

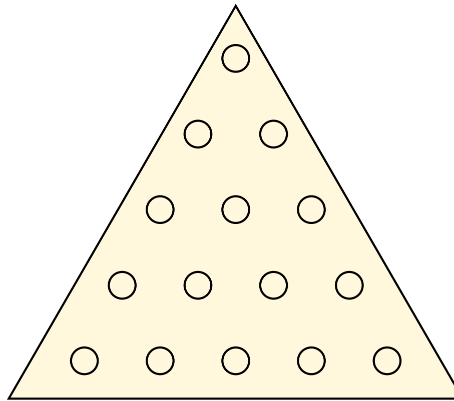
13.3. AcroTEX Peg Solitaire

The following is an example of the use of the Action tab for check boxes. It is available on my AcroTEX web site.

AcroTEX Peg Solitaire

Instructions

1. To begin, click a peg to remove it from the game board.
2. Play the game by jumping over a peg and into an empty hole.
3. To make a jump, click a peg, it will turn red, then click an empty hole. The red peg and the empty hole must be horizontally or diagonally aligned with a single peg in between.
4. If the jump is valid, the jump occurs, just as in checkers.
5. The game is over when there are no more valid jumps possible.



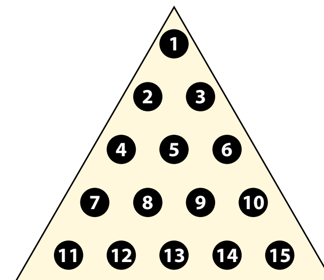
History

Scoring

If you leave…

1. three pegs: Score 10 points
2. two pegs: Score 25 points
3. one peg: Score 50 points
4. one peg in the hole initially left empty: Score 100 points
5. one peg in one of the center holes (holes 5, 8 or 9): Score 150 points
6. eight pegs with no further jumps possible: Score 200 points

Legend



Well, that's pretty much it for now, I simply must get back to my retirement. DPS

www.AcroTeX.net
AcroTeX PDF Blog