



AcroTeX.Net

AcroTeX PDF Blog

Processing Acrobat Forms using JavaScript

Internal Processing of a Field

Part 2: The Event Object for Processing

Text & Choice Fields

D. P. Story

Table of Contents

14 The Event object for Text and Choice Fields	3
14.1 The Chain of Events	3
14.2 Summary of Events and Examples	7

14. The Event object for Text and Choice Fields

In addition to the General, Appearance, Options, and Action tabs, text fields and editable combo boxes have the Format, Validate, and Calculate tabs. This article concerns the chain of events that occur when a user uses keystrokes or a paste operation to input text into a text field or editable combo box.

Blog Convention: Before continuing, let us agree that when I make a statement about a *text field* within the context of the event object and internal processing of fields, I mean, unless otherwise specified, to include the *editable combo box*. I just get tired of typing “editable combo box,” there I’ve done it again.

When the user enters data (either through keyboard input or by a paste operation), and commits the data, a chain of events is triggered; these events are Keystroke, Validate, Calculation, and Format, and correspond to the tab names of the properties dialog box of the field.

14.1. The Chain of Events

For completeness sake, we list the chain of events associated with entering data into a text field (or an editable combo box). The following is the list of events that occur, and in the order they occur, as they are documented in [Form event processing](#).¹

- **Keystroke Event:** Each time the user enters data into a text field (or editable combo box), a Keystroke event is triggered. Script that are executed when a Keystroke event occurs are entered through the Format tab. Select Custom from the Select Format Category, and you will see windows into the Custom Keystroke Script editor (and the Custom Format Script editor).

The Field/Keystroke event object Properties: The properties listed below are defined during the Keystroke event, as quoted from [Field/Keystroke](#) of the *JavaScript for Acrobat API Reference*:

- The target for this event is the field whose keystroke script is being executed. This event also defines the `commitKey`, `change`, `changeEx`, `keyDown`, `modifier`, `selEnd`, `selStart`, `shift`, `targetName`, `value`, and `willCommit` properties.
- This event listens to the `rc` return code. If set to `false`, the keystroke is ignored. The resulting change is used as the keystroke if the script wants to replace the keystroke code. The resultant `selEnd` and `selStart` properties can change the current text selection in the field.

¹If a document has one or more fields with calculation scripts, some events may appear to be out of order. See description of the Calculation and Format events below.

Data Committed. The user commits the data by either clicking outside the field, tabbing to another field, or pressing the enter button. When the data is committed, this initiates a series of cascading events.

- **Keystroke Event (Part II):** After the data is committed, the `event.willCommit` property is set to `true`, and there is a final call to the Keystroke script. If any special pre-commit processing is needed, it can be done at this time. The Keystroke script can be written as a conditional:

```
if ( event.willCommit ) {
    <Keystroke script after commit>
    ...
} else {
    <Keystroke script>
    ...
}
```

DPS Comments on Keystroke:

- Keystroke values:
 - * During the Keystroke event, the `event.value` property is *readonly*.
 - * Pre-commit: At the instant the user enters a keystroke (or completes a paste operation), `event.value` is the string that the user has previously entered; `event.change` is the keystroke (or pasted string) just entered.

If the field is a combo box (or list box), the selection of a list item is implemented as a Keystroke event: `event.change` is the *face* (or *appearance*) value of the list item, and `event.changeEx` is the *export value* of the list item. If the combo box is editable and the user inputs custom text, the keystrokes are handled the same way as input into a text field.

The property `event.change` is both read and write. The script can change the user's input in the Keystroke event. For example,

```
event.value=event.value.toUpperCase();
```

changes the user keystrokes to upper case.

The property `event.changeEx` is read access only.

The user's keystroke can be rejected by putting `event.rc=false`, see comments on `event.rc` below.

Beginning with Acrobat 6.0, `event.changeEx` is defined for text fields. When `event.fieldFull` is `true`, `changeEx` is set to the entire text string the user attempted to enter and `event.change` is the text string cropped to what fits within the field. For additional details, see the description of the `fieldFull` property.

- * **Commit:** After the user has committed the data, the keystroke script will be executed once more, at that time, `event.value` is the entire text entered by the user. The script writer has a chance to scan the string once more before the Validate event begins.

`event.value` is read/write, but setting `event.value` does not change the string the user has input. The value of `event.value` is passed on to the Validate script as the value of `event.value`. It is this value that the Validate script sees. Normally, `event.value` is not set in the Keystroke script.

When `event.willCommit=true`, the entire input of the user can be rejected by the Keystroke script by putting `event.rc=false`, see comments on `event.rc` below.

- The `event.rc` property: Within the Keystroke script, `event.rc=false` cancels the latest keystroke (or paste operation) when `event.willComment=false`; and cancels out *the entire user input* (since the last commit) when `event.willCommit` is `true`. After cancellation, the new value of the field is the current field value (`event.target.value`), the value of the field when the user brought focus to the field, or the value of the user's last *validated* input, whichever occurred most recently.
- **Validate Event:** This is the first event that is generated after the user commits the data during the Keystroke event. The Validate script may be used to verify that the value entered was correct, meets certain conditions that could not have otherwise be verified earlier in the chain of events. If the Validate event is successful, the next event triggered is the Calculate event.

The Field/Validate event object Properties: The properties listed below are defined during the Validate event, as quoted from [Field/Validate](#) of the *JavaScript for Acrobat API Reference*:

- The `target` for this event is the field whose validation script is being executed. This event also defines the `change`, `changeEx`, `keyDown`, `modifier`, `shift`, `targetName`, and `value` properties.
- This event does not listen to the `rc` return code. If the return code is set to `false`, the field value is considered to be invalid and the value of the field is unchanged.

DPS Comments on Validate:

- `event.value` is both read and write and is the value passed to it by the Keystroke event. The user input can be changed from within the Validate script.
- By putting `event.rc=false`, the *entire user input* can be made invalid, the the old field value is then used as the field value. The "old field value" is the one that was in effect when the user brought the field into focus, or the value of the user's last *validated* input, whichever occurred most recently.

- The properties `event.change` and `event.changeEx` both equal to the empty string, these two seem to be of little use.
- **Calculate Event:** The event that follows the Validate event. This event is defined when a change in a form requires that all fields that have a calculation script attached to them be executed. All fields that depend on the value of the changed field will now be recalculated. These fields may in turn generate additional Field/Validate, Field/Blur, and Field/Focus events.

The *calculation order array* contains an ordered list of all fields that have a calculation script attached. In addition to the user interface (see the Edit Field menu under the Forms menu while in Form editing mode), the calculation order can be changed through the `Field.calcOrderIndex` property.

The Field/Calculate event object Properties: The properties listed below are defined during the Calculate event, as quoted from `Field/Calculate` of the *JavaScript for Acrobat API Reference*:

- The target for this event is the field whose calculation script is being executed. This event also defines the `source` and `targetName` properties.
- This event listens to the `rc` return code. If the return code is set to `false`, the field's value is not changed. If `true`, the field takes on the value found in the `value` property.

DPS Comments on Calculate: The property `event.value` is both read and write, the value of `event.value` is then the (new) field value. If `event.rc` is set to `false` within the Calculation script, then any value assigned to `event.value` will be invalid, and the current field value (`event.target.value`) remains as the value of the field. The Calculate event occurs if `event.rc=true` from the Validate event. When the Calculate event begins, the current field value is the one validated by the Validation script in the previous Validate event.

When the Calculate event set the value of `event.value`, that triggers a Keystroke event (with `event.willCommit=true`), followed by Validate and Format (twice). A change in the value at this stage must be validated and formatted.

- **Format Event:** This event is triggered once all dependent calculations have been performed. The Format event allows the attached JavaScript to change the way that the data value appears to a user (also known as its *presentation* or *appearance* value). For example, if a data value is a number and the context in which it should be displayed is currency, the formatting script can add a British pound sign (£) to the front of the value and limit it to two decimal places past the decimal point.

The Field/Format event object Properties: The properties listed below are defined during the Format event, as quoted from `Field/Format` of the *JavaScript for Acrobat API Reference*:

- The target for this event is the field whose format script is being executed. This event also defines the `commitKey`, `targetName`, and `willCommit` properties.
- This event does not listen to the `rc` return code. However, the resulting `value` is used as the field's formatted appearance.

DPS Comments on Format: The property `event.value` has a meaning different from its meaning for the Keystroke and Validate events. Within the Format script, `event.value` is the *presentation* or *appearance* value. The value of `event.value` does not change the field value. For example, the code

```
event.value="\u00a3"+event.value;
```

places the British pound symbol in front of value of the field. This does not change the value of the field, it only changes the *appearance* of the value. To see this, blur the field, then focus it again, the value of the field will appear, blurring the field or pressing the Enter key will cause the Format script to execute again.

Important: When a page is opened containing a Format script, Acrobat/AR executes that script to format the value of that field.²

Table 1 on page 11 summarizes the chain of events, and the important `event.change` and `event.value` properties.

14.2. Summary of Events and Examples

When the user enters data (either through keyboard input or by a paste operation), the current field value is *cached* (saved) and becomes readonly. The current field value can still be accessed through the `event.target.value` property. This value remains readonly until the Calculate event occurs, or, if there is no Calculate event, until the Format event occurs, at which time, the new value, as input by the user, becomes the current field value, or the cached value is restored as the field value if the chain of events is cancelled by setting `event.rc=false` in the Keystroke (when `event.willCommit=true`), Validate, and Calculation (if the value of the field is changed in by this script) events.

There are four distinct “values” of a text field, *as the user enters text into the field*:

1. **The Current Field Value:** The value of the field before the user began to enter data. This value is readonly when the user begins entering text, and can be accessed by the script writer through the `event.target.value` property.

²This behavior is often used by those in academics who use Ghostscript to produce PDF. Ghostscript does not support document level JavaScript, so authors define functions in a Format script of a text field. When Acrobat/AR executes the Format script, the functions become defined document-wide. (There is no requirement, by the way, that a Format script must format the value of the field.)

2. **The Ongoing Field Value:** The value of the field the user sees just before a keystroke input or a paste operation into the field. This value is accessed through the event .value property. After the user commits the data, the event .value is the text string entered by the user.
3. **The Current Data Entered Value:** When the user presses a key for keyboard input or executes a paste operation into the field, the new data enters the field. The value of this new data is accessed through the event .change property.
4. **The Appearance Value:** This value determines how the value of the field is displayed. It is accessed through the event.value property during the Format event.

The values described in (2) and (3) are the two most important values the script writer needs to handle as the user enters data into the text field.

While my head (and your head) is still swimming from that mass of obscure facts and observations, let's to to the examples.

Example 14.1. Below is a text field and an editable combo box. Play around with them by entering a few keystrokes and pressing the enter key (clicking outside the field, or tabbing out of the field) to commit the data.³ Open the JavaScript Debugger Console window, all output is written to that window. If you are using Adobe Reader, click [open my console](#).

As you enter data, observe the events as they occur in the console window. Press enter, to commit the data, see the Keystroke event occur once more, followed by the Validate event and the Format event. These fields do not have Calculation script, so that event does not appear.

Points to Observe: Keep track of the values of the field value (event.target.value), event.change, and event.value. Note that event.target.value=event.value for the first time in Format event (and Calculate event, not shown). In the Format script, we format the value of the field by `if(event.value!="") event.value="Blog #21: " + event.value`. We have a conditional to avoid formatting the empty string.

As a secondary goal, in the Keystroke event observe the values of event.selStart and event.selEnd. Experiment by copy and pasting text into the field, either by insertion, or highlighting some of the text and pasting text in to replace the highlighted text. In particular, note that when you select an item from the drop down menu of the combo box, event.selStart=event.selEnd=-1, I've used this fact to detect when a user has selected a list item, rather than entering custom text. □

We next present an example for the Calculation event, where one event is the sum of the other two. You can trace the chain of events as you enter numerical data.

³Pressing enter commits the data without losing focus (blurring the field), the other two methods blur the field in addition to committing the data.

Example 14.2. Calculate. In the top two text fields you can input numbers. The bottom field is readonly and its value is calculated as the sum of the top two. I've simplified what is written to the [console window](#). Observe the sequence of events as you enter and commit data into fields.

Observe also the occurrence of the Format event. Recall the description of the Format event above, it says, in part, "This event is triggered once all dependent calculations have been performed." As you examine the console window, you can see how this plays out. □

Since the original publication of Blog #21, I realized that `event.value` is both read and write from within the Keystroke event, when `event.willCommit` is true, but the behavior if the chain of events is inconsistent, and may be a bug. Here's an example to illustrate.

Example 14.3. Enter a short string into the text field and commit it. In the Keystroke event, we have `if (event.willCommit)event.value="PDF Blog"`. There are three cases to consider.

1. **Set event.value when there is a Validate script.** The Validate script does not set `event.value`.

In the [console window](#), you'll see...

```
Keystroke(willCommit): event.value=PDF Blog
Validate: event.value=PDF Blog
Calculate: event.value=<your text>
Format: event.value=<your text>
```

When you set `event.value` within the Keystroke script (when `willCommit` is true), this value *does not affect* the user input. This value *is passed on* to the Validate event, but *is not passed on* to either the Calculate event or the Format event.

2. **Set event.value when there is no Validate script.** No Validate script at all. Results are shown in the [console window](#):

The value set in the keystroke script is visible in the text field, as is seen below.

```
Keystroke(willCommit): event.value=PDF Blog
Calculate: event.value=PDF Blog
Format: event.value=PDF Blog
```

The setting of `event.value` in the Keystroke event takes root in this case.

3. Set event.value when there is a validate script, and this validate script also sets event.value.

Within the Validate script, you can set the value of `event.value`, and if `event.rc=true`, this value becomes the new field value. Here is an illustration of this last statement; in the Validate script we set `event.value` with `event.value=event.value`.

The output to the console window is given below.

```
Keystroke(willCommit): event.value=PDF Blog
Validate: event.value=PDF Blog
Calculate: event.value=PDF Blog
Format: event.value=PDF Blog
```


By setting `event.value=event.value`, the behavior of this field is the same as in Case #2.

The behavior of the event chain in Case #1 is significantly different from that of Cases #2 and #3. The behavior in Case #1 must surely be a bug. □

Now let's summarize while everything is still fresh in our minds. Suppose we have a text field whose value is "Acro" and we enter the keystrokes "T", "e", "X". The Format script is `event.value="Blog #21: "+event.value`. The results are shown in [Table 1](#) on page 11.

That's good stuff!

I had more planned for this article, but I think I'll move that material to AcroTeX Blog #22, this will give you time to play around with these examples for the purpose of understanding the chain of events. Study this material well, there will be a quiz in Blog #22.

In the next articles, we'll see less theory and more examples. Well, that's pretty much it for now, I simply must get back to my retirement. 

Event	Input	change ^a	Field value ^{b c}	value
Keystroke	T	T	Acro	Acro ^d
Keystroke	e	e	Acro	AcroT ^d
Keystroke	X	X	Acro	AcroTe ^d
Keystroke (commit)			Acro	AcroTeX ^e
Validate			Acro ^f	AcroTeX ^g
Calculate			AcroTeX ^h	AcroTeX ⁱ
Format			AcroTeX	Blog #21: AcroTeX

^aevent . change is both read and write during a keystroke event, when event . will-Commit=false.

^bThis value is event . target . value.

^cField value: This property is readonly, trying to set it will cause an exception to be thrown.

^devent . value: This property is readonly, no exception is thrown, however, if you try to set this value.

^eevent . value is both read and write, but setting this property may not change the field value. The value is passed to the Validate event. See Example 14.3 for details.

^fField value: After the commit of Keystroke, if event . rc=true, the field value becomes "AcroTeX" in the Calculate/Format events, if event . rc=false, the sequence of events terminates, and field value remains as "Acro".

^gevent . value is both read and write. Setting event . value, if event . rc is true, changes the field value. Putting event . rc to false in the Validate event, invalidates user input, and terminates the event chain.

^hField value: If event . rc=false in the Validate event, the Calculate event does not occur; otherwise, the new field value is "AcroTeX".

ⁱIf event . rc=false in the Calculate event, then any value set by event . value by the Calculate script is ignored, the Format event executes in any case.

Table 1: Summarizing the Event Properties