AcroTEX.Net

AcroTEX PDF Blog

Processing Acrobat Forms using JavaScript

Internal Processing of a Field

Part 3: The Format and Selection Change Tabs

D. P. Story

Warning: In Sections 15.2–15.6 we discuss the (undocumented) folder-level JavaScript that ship with Acrobat/AR. The ones discussed have all been around since version 5, and the extended versions since version 6. While there is always a desire on the part of the Acrobat engineering team to maintain compatibility with previous versions, changes can occur. By using these functions directly you run the risk—slight as it might be—of incompatibilities with future versions of Acrobat. DPS

Copyright © 2009 dpstory@acrotex.net
AcroTEX PDF Blog #22
http://www.acrotex.net
Published: February 1, 2009

www.AcroTeX.net

Table of Contents

15 Format tab for the Text Field and Editable Combo Box **3**

15.1 Format Category: Custom 4

- Custom Keystroke Script 4
- Custom Format Script 7

15.2 Format Category: Number 8

15.3 Format Category: Percentage 12

15.4 Format Category: Date 14

15.5 Format Category: Time 19

15.6 Format Category: Special 21

16 Keystroke/Format of Combo Box: Editable versus Non-Editable **24**

17 Selection Change for the List Box **29**

AcroTeX PDF Blog

www.AcroTeX.net

15. Format tab for the Text Field and Editable Combo Box

In AcroTEX Blog #20, we investigated the Action tab; in that article, the events of `Mouse Enter`, `Mouse Up`, `Mouse Down`, and `Mouse Exit` are triggered by direct user interaction with the document. In AcroTEX Blog #21, we explored the chain of events—Keystroke, Validate, Calculate, and Format—that occur when a user enters data into a text field or editable combo box, and surveyed the important properties of the `event` object for each of these events.

Each of the events (Keystroke, Validate, Calculate, and Format) can have a script attached to it:

- Keystroke and Format event: Custom scripts are entered through the Format tab; in addition to custom script, there are several built-in scripts that ship with Acrobat/AD, these are available through the Format tab user interface.
- Validate event: A custom script can be entered through the Validate tab; there is also a built-in script for validating common requirements, this script is available through the Validate tab.
- Calculate tab: a custom script can be entered through the Calculate tab; also included in this tab are some built-in scripts for making common calculations, these scripts are available through the user interface of the Calculate tab.

In this article, we study the scripting associated with the Format tab. We shall discuss custom Keystroke and Format scripts, as well as survey the built-in scripts that ship with Acrobat/AD and are available through the user interface.

As was just mentioned above, the Format tab controls the Keystroke event and the Format event. These two events are at opposite ends of the event chain; consequently, it is hard *not to mention*—if only in passing—the other two tabs, which control the scripting for the Validate and Calculate events, because they are so closely intertwined. Calculate and Validate will be taken up in detail in a later AcroTEX PDF Blog.

The rest of this section is organized by the Format category of the Format tab. In the Format tab, the user can Select a format category: None, Number, Percentage, Date, Time, Special, and Custom. We begin with the Custom format category.

When the Custom format category is selected from the Format tab in the Text Field (or Combo Box) Properties dialog box, the Custom Format Script and Keystroke Script editing boxes are revealed, into which you can enter your script, see Figure 1, page 4. The other format categories have underlying Format and Keystroke scripts which are defined in `AForm.js` folder JavaScript file.[1] Though the Keystroke and Format scripts cannot be accessed through the user interface, we shall survey the functions called by these scripts in Sections 15.2–15.6.

[1] This file is now pre-compiled into the binary file named `JSByteCodeWin.bin` for the Windows platform, and `JSByteCodeMac.bin` for the Mac OS.

AcroTeX PDF Blog

Acknowledgements: I extracted some of the information found in Sections 15.2–15.6 by snooping through a PDF file using the PDF CanOpener, by WindJack Solutions, Inc.[2, 3] Section 2.3.6.1 of *The LaTeX Web Companion, Integrating TeX, HTML, and XML*, by Michel Goossens, Sebastian Rahtz, *et al*, was also used as a resource for Sections 15.2–15.6. This book is one of the few that documents (at least partially) the folder level JavaScript functions that are used by the built-in scripts of the Format, Validate, and Calculate tabs. Of course, in this article, we concentrate on the Format tab.

15.1. Format Category: Custom

In this section we illustrate custom Keystroke and Format scripting, all the while trying to avoid Validate and Calculate scripting, which will be discussed in a later blog article.

You enter custom Keystroke or Format script of a text field or an editable combo box by bringing up the properties box, selecting the Format tab, and then selecting the Custom format category. The Custom Format Script and Custom Keystroke Script editing boxes are made available. Click the Edit to enter the script editor. See the figure to the right.

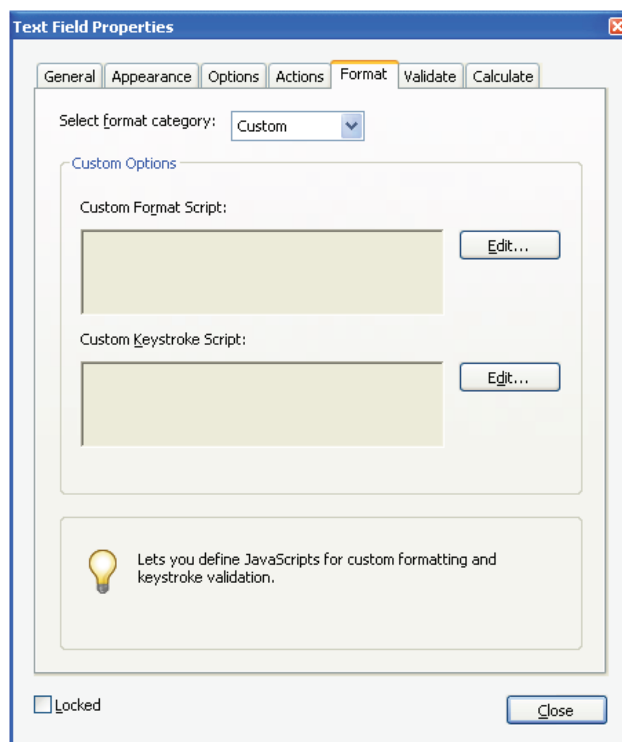

Figure 1: Custom Format Category

• Custom Keystroke Script

As was noted in AcroTeX Blog #21, there are two distinct code segments of the custom Keystroke script:

- Keystroke script before committing, when `event.willCommit=false`
- Keystroke script after committing, when `event.willCommit=true`

[2] `http://www.windjack.com`

[3] I downloaded a fully functional version of the product, an Acrobat plug-in, for a 10 day trial period. I found the PDF CanOpener to be intuitive and very easy to use. It is indeed an excellent tool for examining (and editing) the structure of a PDF file.

www.AcroTeX.net

AcroTeX PDF Blog

www.AcroTeX.net

The basic structure for these to segments is

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

The script itself may contain one or both of these segments, there is no requirement to have a Keystroke at all, for that matter.

Example 15.1. Require all text to be upper case. Enter some text in the text field below. The text will be changed to upper case by Keystroke script.

The Keystroke script for this field is

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

This script will be executed as the user enters text into the field, and when the user commits the text to the field; however, when the user commits, `event.change=""`, so the script really does nothing in this case. □

The next example turned out to be a major project for me. In the Special Format Category, (Section 15.6 on page 21) includes a special Keystroke script for formatting a (U.S.) Zip Code; a good example, I thought, would be to write a similar script but for the *U.K. postcode*. The resources used for this next example include Postal codes in the United Kingdom, an article Wikipedia. A U.K. postcode is an alphanumeric code; such codes generally have the form:

Mask	**Example**	**Comments**
`A9 9AA`	`W3 9SX`	
`A99 9AA`	`M53 0PQ`	
`A9A 9AA`	`Y5A 3DP`	
`AA9 9AA`	`PQ2 5LQ`	
`AA99 9AA`	`SW12 9SU`	my brother's postcode
`AA9A 9AA`	`SM9W 3DP`	

The letter `A` stands for a letter (A-Z), and `9` stands for a number (0-9). Not all letters are used in the various positions defined above. In addition to these, there is a special postcode, this is `GIR 0AA`, here the `AA` are literally two capital letter `A`s.

The Wikipedia article gives a regular expression for validating these postcodes, this is

AcroTeX PDF Blog

```
(GIR0AA|[A-PR-UWYZ]([0-9]{1,2}|([A-HK-Y][0-9]|
    [A-HK-Y][0-9]([0-9]|[ABEHMNPRV-Y]))|
    [0-9][A-HJKS-UW]) [0-9][ABD-HJLNP-UW-Z]{2})
```

In the above regular expression, I've broken the expression to fit the width of the page. Note that the allowable alphabetic characters are listed in various character classes.

There are a couple of ways to implement this validation regular expression. The first example is very simple, the second one is considerably more extensive.

Example 15.2. A simple custom Keystroke script for validating a U.K. postcode.

Enter a U.K. postcode:

The custom Keystroke script is given below, where again, I've broken the long regular expression across several lines.

```
1  if (event.willCommit) {
2      if (event.value !="") {
3          var ukpcRe1=/^(GIR0AA|[A-PR-UWYZ]([0-9]{1,2}|
4              ([A-HK-Y][0-9]|[A-HK-Y][0-9]([0-9]|[ABEHMNPRV-Y]))|
5                  [0-9][A-HJKS-UW]) [0-9][ABD-HJLNP-UW-Z]{2})$/;
6          if ( !ukpcRe.test(event.value) ){
7              app.beep(0); event.rc=false;
8          }
9      }
10 } else
11     event.change=event.change.toUpperCase();
```

Code Comments: This script changes keystrokes to upper case (line 12); after the user has committed the text, we check the user input against the regular expression `ukpcRe1`. □

The above example is certainly functional and lets the user enter a valid U.K. postcode. If the code is not valid, including the required space, the entry is rejected. The script gives no hint as to the problem with the postscode entered, perhaps a typo on the user's part, or an invalid letter was entered in one of the alphabetic positions.

The next example is much more elaborate. It tries to guide the user as he/she enters the code, it detects any error as it is entered. (At least we hope!)

Example 15.3. A comprehensive custom Keystroke script for validating a U.K. postcode. As the user enters a postal code, what has been entered is compared against certain "partial entry" regular expressions, These RegExps were derived from the master RegExp provided by the Wikipedia article. If what the user has entered does not match up against any of these "partial entry" RegExps, the `event.rc` is set to `false`.

www.AcroTeX.net
AcroTeX PDF Blog

www.AcroTeX.net

Try entering postal codes that follow the any of the patterns below; also, deliberately enter a number when there should be a letter, or a letter when there should be a number. Some letters may be flagged as invalid because not all letters are used in each of the positions designated for letters. Below is the table of masks given earlier, with examples.

Mask	Example	Comments
`GIR 0AA`	`GIR 0AA`	special case
`A9 9AA`	`W3 9SX`	
`A99 9AA`	`M53 0PQ`	
`A9A 9AA`	`Y5A 3DP`	
`AA9 9AA`	`PQ2 5LQ`	
`AA99 9AA`	`SW12 9SU`	my brother's postcode
`AA9A 9AA`	`SM9W 3DP`	

Enter a U.K. postcode:

This text field calls the custom Keystroke function `ukpcKeystroke()` defined as document-level JavaScript. See the script titled **UK Postal Codes** in the menu system under `Advance > Document Processing > Document JavaScripts`, for those who have a flavor of Acrobat. I've included some documentation comments in the script for your review.

The JavaScript function `AFMergeChange`, which is defined in `AForm.js` and is distributed with the Acrobat and AD products, is used in this example. Though officially an undocumented function, it appears in the *JavaScript for Acrobat API Reference*, see the example following the description of the `event.selEnd` property. The function merges the value of `event.value`, the value of the text field that the user has already entered, with the value of `event.change`, which is the text the user is currently entering. The function takes into consideration that the user may not enter (or paste) text at the end of the current text. This is a very handy function for writing custom Keystroke script. □

• Custom Format Script

The Format event comes at the tail end of the event chain, after the entry is committed and validated. Within the Format event, the property `event.value` holds the (default) *appearance value* of the field (which is just the current value of the field). Setting `event.value` does not change the value of the field, only its appearance. Below is a simple example.

Example 15.4. Formatting a Field: Spreading the text. Enter text into the field below, and it will be formatted.

When you enter and commit your text, the field is formatted; the letters are spread. What you

AcroTeX PDF Blog

see is the *appearance value* of the field, not the value itself. When you click into the field and bring the field back into focus, you'll see the value you entered into the text field; this value is the value of the field.

The script for this Format script is very simple:

```
if (event.value!="")
    event.value=event.value.replace(/(\B|\s)/g,"$1 ");
```

Here, we use a regular expression `/(\B|\s)/g` to replace any non-word boundary or space character with what is matched `$1` followed by a space. The custom Format script above does not spread out punctuation marks, or other characters that are not letters. A more complex regular expression is needed. This is left as an exercise for you. □

15.2. Format Category: Number

This format category is implemented with Keystroke and Format scripts. The Keystroke script calls the folder JavaScript function `AFNumber_Keystroke()` with various parameters, and the Format script calls `AFNumber_Format()`. A field designated as a Number field accepts only numbers into the field, and formats the committed value according to the preferences specified through the user interface.

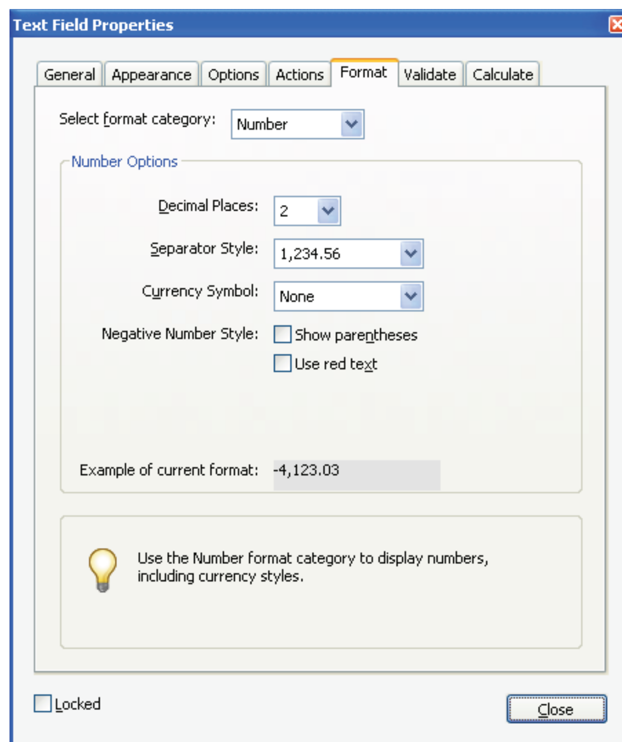

Figure 2: Number Format Category

To select the Number format category of a text field or an editable combo box, bring up the properties box, select the Format tab, then select the Number format category. The Number Options appear, see the figure to the right. Through the options, you can set the number of decimal places, the separator style (set the decimal point symbol and the block number separator), the currency symbol (if any), and how negative numbers are to be presented.

The parameters for `AFNumber_Keystroke()` and `AFNumber_Format()` are given below:

```
AFNumber_Keystroke(nDec,sepStyle,negStyle,currStyle,
    strCurrency,bCurrencyPrepend)
```

www.AcroTeX.net
AcroTeX PDF Blog

www.AcroTeX.net

```
AFNumber_Format(nDec,sepStyle,negStyle,currStyle,
    strCurrency,bCurrencyPrepend)
```

`AFNumber_Keystroke()` sets the value of `event.rc` and, on commit, sets the value of `event.value`, while `AFNumber_Format()` sets the value of `event.value`, which is the appearance value of the field.

The description of the parameters:

- `nDec`: The number of decimal places to appear in the formatted value. `nDec` is a non-negative integer.
- `sepStyle`: The separator style to be used. The separator style determines the character to be used for the decimal point and the character to be used as the number block separator. Possible values of `sepStyle` are 0, 1, 2, 3:

sepStyle	Example	sepStyle	Example
0	1,234.56	2	1.234,56
1	1234.56	3	1234,56

- `negStyle`: The negative number style. possible values are 0, 1, 2, 3:

negStyle	Example	negStyle	Example
0	-1234	2	(1234)
1	1234	3	(1234)

- `currStyle`: This parameter is not used, and is set to 0.
- `strCurrency`: A string representing the currency symbol. The string value of this parameter is the unicode representing the desired currency symbol. The following currency symbols are available through the user interface.

Currency	Unicode	Currency	Unicode
Dollar	\u0024	DM	\u0020\u0044\u004d
Euro	\u20ac	Guilder	\u0066\u006c
Franc	\u0020\u0046	Kroner	\u0020\u006b\uu072
Lira	\u004c\u002e\u0020	Peseta	\u0020\u0050\u0074\u0073
Pound	\u00a3	Yen	\u00a5

  Unicode need not be used, if you pass `"$"` as the value of the `strCurrency` parameter, the Dollar currency symbol is used. If no currency symbol is desired, pass the empty string for the `strCurrency` parameter. For example, using

  ```
  AFNumber_Keystroke(0,0,0,0,"",true);    // custom Keystroke script
  AFNumber_Format(0,0,0,0,"",true);       // custom Format script
  ```

  forces an integer input, with no currency symbol. Most certainly, you are free to use another currency symbol, or, for that matter, any character or sequence of characters.

AcroTeX PDF Blog

www.AcroTeX.net

- `bCurrencyPrepend`: The prepend currency symbol. If `true`, the currency symbol is placed at the beginning of the number; if `false`, the currency symbol is placed at the end of the number. A table of supported currencies follow:

`bCurrencyPrepend`	**Currencies**
`true`	Dollar, Euro, Guilder, Lira, Pound, and Yen
`false`	DM, Franc, Kroner, and Peseta

Example 15.5. Creating a number field with Euro currency symbol. In this text field, we create a number format with two decimal places, the number is red when negative. The Euro currency symbol is used.

The font used must include the currency symbol; here, we use MyriadPro-Regular to match the font of this document.

The Keystroke script is

```
AFNumber_Keystroke(2,0,1,0,"\\u20ac",true);
```

and the Format script is

```
AFNumber_Format(2,0,1,0,"\\u20ac",true);
```

Note that within a JavaScript string, the unicode escape sequence must be double escaped, like so `"\\u20ac"`. □

Important: The built-in Keystroke, Format, Validate, and Calculate functions can be used in your own custom scripts; however, *it is important not to use the names of these functions, or to use a function name with these names as a substring at the top level*. Use these built-in functions within another function. I believe Acrobat does a string search for these built-in function names (such as `AFNumber_Keytroke`), if it finds one, it assumes the script is one of the built-in routines; consequently, it does not treat the script as a custom script and does not present the editing interface to the script.

The next example illustrates the user of `AFNumber_Keystroke` and `AFNumber_Format` within larger custom scripts.

Example 15.6. Extending the use of AFNumber_Keystroke and AFNumber_Format. In this example, we allow a number input with at most two decimal places. After the value is committed, we calculate the value of the greatest integer function (the terminology used in Mathematics, this is the floor function, the term used in Computer Science).[4] We format the

[4]The value of the greatest integer function of x is $[x]$, the greatest integer less than or equal to x.

AcroTeX PDF Blog

www.AcroTeX.net

final appearance using standard Mathematical notation for the greatest integer function, which uses square brackets; for example [3.14]=3 and [-88.17]=-89.

Enter a number: period (.) is the decimal point

comma (,) is the decimal point

The script for this field is short enough to present here, and comment on. The custom Keystroke and Format script simply calls the functions `myNumber_Keystroke` and `myNumber_Format`, respectively. These functions—defined at the document level—are given below.

Keystroke Script Function Call. The custom Keystroke script for the above text field is

```
myNumber_Keystroke(2,1,0,0,"",true)
```

the function `myNumber_Keystroke` is listed below.

```
1  function myNumber_Keystroke(nDec,sepStyle,negStyle,currStyle,
2      strCurrency,bCurrencyPrepend) {
3      AFNumber_Keystroke(nDec,sepStyle,negStyle,currStyle,
4          strCurrency,bCurrencyPrepend);
5      if (!event.willCommit) {
6          if (event.change.replace(/\s*/g,"")=="") return;
7          var value=AFMergeChange(event);
8          var nPos=String(value).indexOf((sepStyle<2) ? ".":",");
9          if ( nPos==-1 ) return; // do nothing if there is no decimal point
10         var cDec=String(value).substring(nPos+1);
11         if (cDec.length>2) {
12             event.rc=false;
13             app.beep(0);
14             app.alert("Only two decimal places are permitted.");
15         }
16     }
17 }
```

Code Comments. It would be a mistake to name this function `myAFNumber_Keystroke` as the built-in function `AFNumber_Keystroke` is a substring of this function name. The editor would not be available for your use in this case. This function uses the same parameters as `AFNumber_Keystroke`. The first thing we do is to call `AFNumber_Keystroke`, line (3), then, if the entry is not committed, we merge the change, line (7), then search for a decimal point. The decimal point is either a period (.) or a comma (,), depending on the value of `sepStyle`, line (8). We then extract the decimal part, `cDec`, of the number in line (10). If the length of this string is greater than 2, we reject the latest change, lines (11)–(15). In this way, we enforce an at most two decimal place restriction on input.

Format Script Function Call. The custom Format script for the above text field is

```
myNumber_Format(2,1,0,0,"",true)
```

AcroTeX PDF Blog

www.AcroTeX.net

The function myNumber_Format is listed below.

```
1  function myNumber_Format(nDec,sepStyle,negStyle,currStyle,
2      strCurrency,bCurrencyPrepend) {
3      var value=event.value;
4      var nPos=String(value).indexOf(".");
5      if ( nPos !=-1 ) {
6          var cDec=String(value).substring(nPos+1);
7          if (cDec.length<2) nDec=cDec.length;
8      } else nDec=0;
9      AFNumber_Format(nDec,sepStyle,negStyle,currStyle,
10         strCurrency,bCurrencyPrepend);
11     if ( event.value !="")
12         event.value="["+event.value+"] = " + Math.floor(value);
13 }
```

Code Comments. The function myNumber_Format has the same parameter set as has AFNumber_Format. In line (3) we save the unformatted value of the field in the variable value. In line (4) we search for a decimal point (a period). If one is found, line (5), we extract the decimal string; if the length is less than 2, line (7), we put nDec=cDec.length. Here, if the user enters no decimal place, then no decimal point will be shown in the appearance of the value; if the user uses only one decimal point, then we format only one decimal point. If the user does not enter a decimal point, line (8), no decimal point while be used in the appearance value. nDec is the parameter passed to AFNumber_Format. In line (9) we call AFNumber_Format with the dynamic first parameter cDec. Finally, in line (12), we format the appearance value using correct Mathematical notation, the greatest integer function is computed using Math.floor(value). We use value as the parameter of Math.floor, because value is a proper number, not a string with a comma as its decimal point, as would be the case when sepStyle > 1. □

Note: The functions AFNumber_Keystroke and AFNumber_Format have been around since (at least) version 5. For compatibility, these function will continue to be defined so they are safe to use.

15.3. Format Category: Percentage

A text field or editable combo box that is designed to accept and format numbers that are interpreted as a percentage should use the Percentage format category.

To select the Percentage format category of a text field or an editable combo box, bring up the properties box, select the Format tab, then select the Percentage format category. The Percentage Options appear, see Figure 3, page 13. Through the options, you can set the number of decimal places and the separator style (set the decimal point symbol and the block

AcroTeX PDF Blog

number separator).

The Percentage format category, see Figure 3, is implemented with Keystroke and Format scripts. The Keystroke script calls the folder JavaScript function `AFPercent_Keystroke()` with various parameters, and the Format script calls `AFPercent_Format()`. A field designated as a Percent field accepts only numbers into the field, and formats the committed value as a percentage; thus, a value of `0.25` is formatted as 25%.

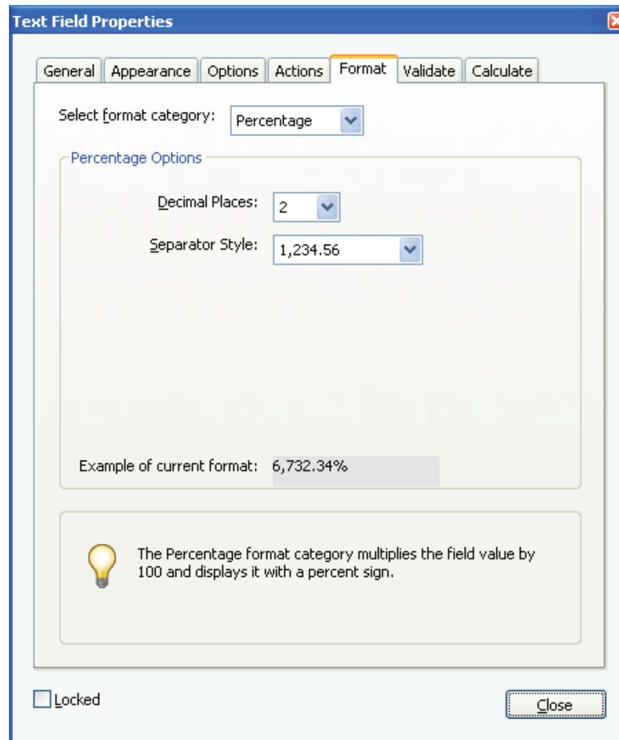

Figure 3: Percentage Format Category

The parameters for `AFPercent_Keystroke()` and `AFPercent_Format()` are given below:

```
AFPercent_Keystroke(nDec,sepStyle)

AFPercent_Format(nDec,sepStyle,bPercentPrepend)
```

`AFPercent_Keystroke()` sets the value of `event.rc` and, on commit, sets the value of `event.value`, while `AFPercent_Format()` sets the value of `event.value`, which is the appearance value of the field.

www.AcroTeX.net
AcroTeX PDF Blog

www.AcroTeX.net

The description of the parameters:

- `nDec`: The number of decimal places to appear in the formatted value. `nDec` is a non-negative integer.
- `sepStyle`: The separator style to be used. The separator style determines the character to be used for the decimal point and the character to be used as the number block separator. Possible values of `sepStyle` are 0, 1, 2, 3:

sepStyle	**Example**	sepStyle	**Example**
0	1,234.56	2	1.234,56
1	1234.56	3	1234,56

- `bPercentPrepend`: (Optional) If `bPercentPrepend true`, the percent symbol is placed at the beginning of the number; if `false`, the percent symbol is placed at the end of the number. If the `bPercentPrepend` parameter is not listed in the parameter list, the percent symbol is placed at the end of the number.

Example 15.7. Creating a Percent text field.

Enter a number:

The Keystroke script is

```
AFPercent_Keystroke(2,1)
```

and the Format script is

```
AFPercent_Format(2,1)
```

Custom Percentage scripts can be created using built-in functions `AFPercent_Keystroke()` and `AFPercent_Format()` in the same way as a Number field, as illustrated in Example 15.6 on page 10.

15.4. Format Category: Date

A text field or editable combo box that is designed to accept and format alphanumeric characters that are interpreted as a date should use Date format category.

To select the Date format category of a text field or an editable combo box, bring up the properties box, select the Format tab, then select the Date format category. The Date Options appear, see Figure 3, page 15. The options list a number of date masks to choose from.

From the JavaScript viewpoint, the Date format category is implemented through built-in Keystroke and Format scripts. There are two sets of functions that are used:

1. `AFDate_Keystroke` and `AFDate_Format`: The "original" keystroke and format functions for date fields.

AcroTeX PDF Blog

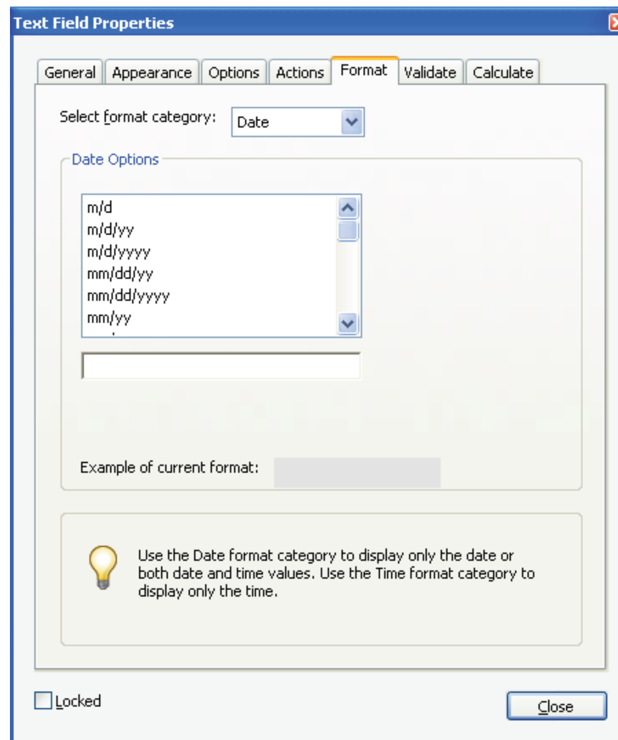

Figure 4: Date Format Category

2. `AFDate_KeystrokeEx` and `AFDate_FormatEx`: The new/extended versions. I'm not sure when these function were first written, I'm guessing version 6.0. I'm still researching this question.

The parameter for the original keystroke and format function is given below. Use these functions to maintain compatibility with all earlier versions of Acrobat/AR.

```
AFDate_Keystroke(pdf)

AFDate_Format(pdf)
```

The description of the parameters:

- `pdf`: The `pdf` parameter is a nonnegative integer and corresponds to an index into a matrix of date formats. The dates are ultimately formatted using the Acrobat JavaScript method `util.printd()` using the selected date format as the `cFormat` parameter of `util.printd()`. A list of permissible values for `pdf`, and the Date formats that correspond to them are shown in Table 1, page 16.

www.AcroTeX.net

AcroTeX PDF Blog

www.AcroTeX.net

pdf	cFormat	**Example**	pdf	cFormat	**Example**
0	m/d	1/7	7	yy-mm-dd	09-01-07
1	m/d/yy	1/7/09	8	mmm-yy	Jan-09
2	mm/dd/yy	01/07/09	9	mmmm-yy	January-09
3	mm/yy	01/09	10	mmm d, yyyy	Jan 7, 2009
4	d-mmm	7-Jan	11	mmmm d, yyyy	January 7, 2009
5	d-mmm-yy	7-Jan-09	12	m/d/yy h:MM tt	1/7/09 2:30pm
6	dd-mmm-yy	07-Jan-09	13	m/d/yy HH:MM	1/7/09 14:30

Table 1: Date formats for AFDate_Keystroke/Format

Example 15.8. Format a date, using pdf=6.

The Keystroke script is

```
AFDate_Keystroke(6)
```

and the Format script is

```
AFDate_Format(6)
```

JavaScript methods are more dynamic than the static methods acquired through the user interface. In the next example, we allow the user to set the date format style through a combo box. We present two methods of dynamically changing the date format, the first using a very simple calculate script, the other uses a hidden field to blur the combo box, details follow.

Example 15.9. Dynamically setting the date format through a combo box using a calculate script.

Enter a date: Choose a format:

Comments: On the quirky behavior of some of the date formats.

- Exercise 1: Enter a month and day, such as July 4; as needed, the current year (or the last year entered) is supplied. The underlying Date functions correctly parses and formats this entry for all date formats.
- Exercise 2: Enter a month, day, and a year, such as July 4, 2009. The following formats are not parsed correctly: mm/yy (removing the day fixes the problem); yy-mm-dd (2009, July 4 works); mmmm-yy (removing the day fixes the problem); mmmm-yy (removing the day fixes the problem).

There are two bugs here: (1) The day is critical, the day must be supplied if and only if the day is specified in the date format; (2) if the year is in the date format, it may be either left off, or if

AcroTeX PDF Blog

www.AcroTeX.net

supplied must be placed in the order indicated by the date format. (Supply the year first, if the format is `yy-mm-dd`.)

Here is how this pair of fields work (together):

The date text field: The following are the scripts for the text field:

Keystroke: Get the current value of the combo box, and feed it to `AFDate_Keystroke`.

```
var f=this.getField("combo15-9");
AFDate_Keystroke(f.value);
```

Format: Get the current value of the combo box, and feed it to `AFDate_Format`.

```
var f=this.getField("combo15-9");
AFDate_Format(f.value);
```

Calculate: Set `event.value` equal to the current value, `event.value`.

```
event.value=event.value;
```

The combo box of date formats: The combo box has no script; however, it should be noted that export values of each of the date formats is the index required by `AFDate_Keystroke` and `AFDate_Format` required for that date format.

Critique: When there is a change in the value of a field, Acrobat updates all fields with calculate scripts. This is the simplest approach, but it does have the additional overhead of introducing a Calculate event. A large number of calculate script can slow Acrobat/AR down quite a bit. □

Example 15.10. Dynamically setting the date format through a combo box using a hidden field.

Enter a date: Choose a format:

Here is how this pair of fields work (together):

The date text field: The following are the scripts for the text field, these are the same as in Example 15.9, except there is no calculate script.

Keystroke: Get the current value of the combo box, and feed it to `AFDate_Keystroke`.

```
var f=this.getField("combo15-10-1");
AFDate_Keystroke(f.value);
```

Format: Get the current value of the combo box, and feed it to `AFDate_Format`.

```
var f=this.getField("combo15-10-1");
AFDate_Format(f.value);
```

The combo box of date formats: Slightly to the right of the combo box is a tiny, transparent push button named `"pb15-10-1tiny"` that is 1 point by 1 point in dimension. This tiny field plays the important role of blurring the combo box without the user being aware of what is happening. It has no attached scripts. Here are scripts of the combo box:

AcroTeX PDF Blog

www.AcroTeX.net

Keystroke: Make the field lose focus (blur it) when the user makes a choice. We blur the field by setting the focus on the tiny push button.

```
if (!event.willCommit) this.getField("pb15-10-1tiny").setFocus();
```

Blur: When the combo box loses focus (blurred), the following script executes. Get the current value of the date field, reset the date field, when place the saved value back into the date field. The resetting of the field is needed so the date field will realize that there is a change in its value when we re-insert the same value.

```
var value=this.getField("txt15-10-1").value;
this.resetForm("txt15-10-1");
this.getField("txt15-10-1").value=value;
```

Putting this code in the Blur script is needed, for by this time, the new value of the combo box is set, and is known to the outside world. When we reset the value of the date field, the Keystroke script then gets the value of the combo box (the one just set), and the rest, as they say, is history.

Critique: This method requires additional overhead of introducing a "dummy field," which adds to the file size. The dummy field must be on the same page as (and close to) the field that is using it for when you set focus, Acrobat/AR will change pages to show the field what is now in focus. □

Additional date formats were introduced in version 6, and the following built-in functions were added.

```
AFDate_KeystrokeEx(cFormat)
AFDate_FormatEx(cFormat)
```

The functions `AFDate_Keystroke` and `AFDate_Format` just pass their formats, as selectd by index, to their extended counterparts.

The description of the parameters:

- `cFormat`: The use interface lists 24 date formats, plus a field for entering a custom date format. The dates formats are ultimately formatted using the Acrobat JavaScript method `util.printd()` using the selected date format as the `cFormat` parameter of `util.printd()`. These are given in Table 2 on page 19.

  In addition to the 24 formats listed in Table 2, there is a field in the user interface for entering a Custom date format; in this case, execute the built-in functions

  ```
  AFDate_KeystrokeEx(cFormat)
  AFDate_FormatEx(cFormat)
  ```

  using the custom format `cFormat` as the Keystroke and Format scripts, respectively.

AcroTeX PDF Blog

cFormat	Example	cFormat	Example	cFormat	Example
m/d	1/7	d-mmm-yyyy	7-Jan-09	mmmm-yy	January-09
m/d/yy	1/7/09	d-mmm-yyyy	7-Jan-2009	mmmm-yyyy	January-2009
m/d/yyyy	1/7/2009	dd-mmm-yy	7-Jan-09	mmm d, yyyy	Jan 7, 2009
mm/dd/yy	01/07/09	ddd-mmm-yyyy	07-Jan-2009	mmmm d, yyyy	January 7, 2009
mm/dd/yyyy	01/07/2009	yy-mm-dd	09-01-7	m/d/yy h:MM tt	1/7/09 2:30 pm
mm/yy	01/09	yyyy-mm-dd	2009-01-7	m/d/yyyy h:MM tt	1/7/2009 2:30 pm
mm/yyyy	01/2009	mmm-yy	Jan-09	m/d/yy HH:MM	1/7/09 14:30
d-mmm	7-Jan	mmm-yyyy	Jan-2009	m/d/yyyy HH:MM	1/7/2009 14:30

Table 2: Table of Extended Date Formats

Example 15.11. Using the built-in Date formats. In this example, we use one of the formats available through the user interface; we use ddd-mmm-yyyy, which is a format not included in the listing in Table 1, page 16.

Enter a date:

The Keystroke script is

```
AFDate_KeystrokeEx("ddd-mmm-yyyy")
```

and the Format script is

```
AFDate_FormatEx("ddd-mmm-yyyy")
```

A custom date field may be created with the same two built-in functions, just create your own date format, using the rules described the documentation of util.printd() in the *JavaScript for Acrobat API Reference*. Don't forget to fully test your now format! □

15.5. Format Category: Time

Though some of the date formats given in the previous section contain time elements, you may want to create a purely Time formatted text field. The user interface to the Time format category is shown in Figure 5 on page 20. As you can see, there are four pre-defined time formats, and an opportunity to enter you own Custom format.

As before, there are two sets of built-ins, the first two, which take an index parameter into an array of time formsts, and one that takes a time format are its parameter. This latter one probably came in with version 6. Looking at the source code for AFTime_Keystroke(ptf), we see that the parameter pdf is not actually used; it is only used with AFTime_Format(ptf). Therefore, you can use either the original pair AFTime_Keystroke and AFTime_Format, or AFTime_Keystroke (with any parameter) and AFTime_FormatEx. Use AFTime_FormatEx to format a Custom time format.

www.AcroTeX.net
AcroTeX PDF Blog

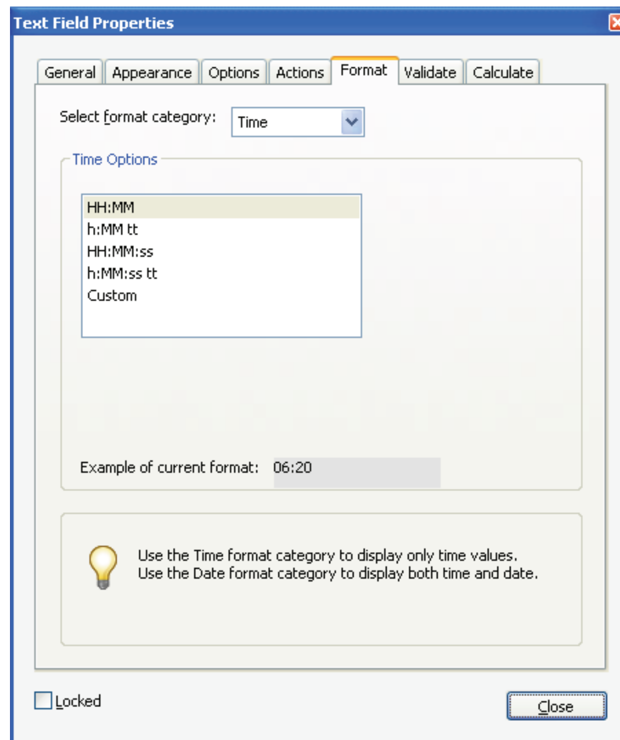

Figure 5: Date Format Category

```
AFTime_Keystroke(ptf)   % pointer time format ptf is not used
AFTime_Format(ptf) % event.value = util.printd(cFormats[ptf], date);
AFTime_FormatEx(cFormat)  % used for custom
```

When using `AFTime_Keystroke` and `AFTime_Format`, the pdf parameter and their corresponding time formats are given in Table 3, page 20.

ptf	cFormat	**Example**	ptf	cFormat	**Example**
0	HH:MM	18:26	2	HH:MM:ss	18:27:41
1	h:MM tt	6:27 pm	3	h:MM:ss tt	6:28:19

Table 3: Time formats for `AFTime_Keystroke/Format`

A Custom format can be obtained by a direct appeal to `AFTime_FormatEx`.

Example 15.12. Creating a built-in time field. In this example, we pass an argument of ptf=1 to `AFTime_Keystroke` and `AFTime_Format`.

Enter a time:

www.AcroTeX.net
AcroTeX PDF Blog

www.AcroTeX.net

In this example, I've also supplied a push button that will populate the time field with the current time.

The Keystroke script is

```
AFTime_Keystroke(1)
```

and the Format script is

```
AFTime_Format(1)
```

Example 15.13. Creating a Custom time format text field.

Enter a time:

The Keystroke script is

```
AFTime_Keystroke(0)
```

and the Format script is

```
AFTime_FormatEx("h:MM (tt)")
```

The `Mouse Up` script for the push button is

```
var cTime=util.printd("h:MM tt",new Date());
var f=this.getField("txt15-13")=cTime;
```

15.6. Format Category: Special

The special category is used for setting up fields that accept patterns of alphanumeric characters, such as a zip code, a phone number or a SSN. The user interface for the Special format category is shown in Table 6 on page 22. The user interface allows the user to enter an Arbitrary Mask in additional to one of the built-in patterns.

The patterns available through the user interface (see Table 6, page 22) use the built-in function `AFSpecial_Keystroke` and `AFSpecial_Format`; while the Arbitrary Mask option is implemented by `AFSpecial_KeystrokeEx`, this option does not have a Format script. `AFSpecial_Format` ultimately formats the pattern using `util.printx()` in the *JavaScript for Acrobat API Reference*.

The parameters for these functions are given below.

```
AFSpecial_Keystroke(psf)
AFSpecial_Format(psf)

AFSpecial_KeystrokeEx(mask)
```

AcroTeX PDF Blog

www.AcroTeX.net

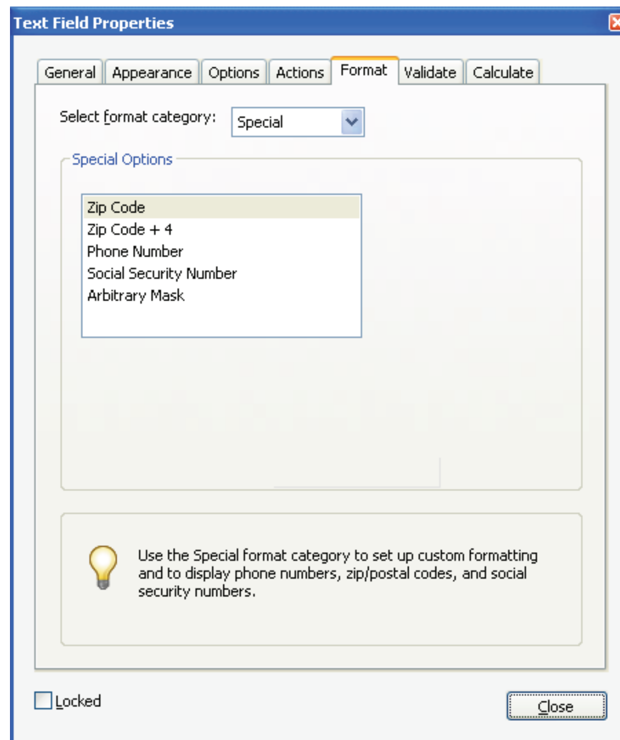

Figure 6: Special Format Category

The description of the parameters:

- `psf`: The `psf` parameter is a nonnegative integer corresponding to each of the patterns available through the user interface. A list of permissible values for `psf` and their meaning is shown in the table below.

psf	**Description**	psf	**Description**
0	Zip Code	2	Phone Number
1	Zip Code + 4	3	SSN

Table 4: Built-in Special Formats

- `mask`: When creating an Arbitrary Mask, `AFSpecial_KeystrokeEx`, defined as folder level JavaScript, is used for a Keystroke script, there is no Format script. The function takes a single argument, `mask`, which consists of alphanumeric characters that have special meaning mixed in with ordinary characters. The special characters are A, X, O, and 9, their meaning is described below.

AcroTeX PDF Blog

www.AcroTeX.net

`A`	Accepts only letters (A–Z, a–z).
`X`	Accepts spaces and most printable characters, including all characters available on a standard keyboard and ANSI characters in the ranges of 32–126 and 128–255.
`O`	The letter "O" accepts alphanumeric characters (A–Z, a–z, and 0–9).
`9`	Accepts only numeric characters (0–9).

Table 5: Special Characters for `mask`

Any other characters that appear in the `mask` must appear literally. There does not seem to be an escape character, so the user cannot be forced to enter a capital letter A or O, or the number 9 in a particular position. This Arbitrary Mask feature seems to be of limited value; it requires input of a fixed length, with a fixed pattern of letters and numbers.

Example 15.14. Creating a Phone Number text field.

Enter a phone number:

The Keystroke script is

```
AFSpecial_Keystroke(2)
```

and the Format script is

```
AFSpecial_Format(2)
```

Example 15.15. Creating an Arbitrary Mask.

Enter the 3-digit security code from the back of your favorite credit card:

The Keystroke script is

```
AFSpecial_KeystrokeEx("999")
```

Example 15.16. Extending AFSpecial_KeystrokeEx in Custom Keystroke The following example uses the premise of entering a Product Key. The key has the form `A9OAA-AO9A9`. Enter a series of numbers/letters that match the Mask.

Please enter the Product Key of your AcroTEX software:

The Keystroke script is implemented as a document JavaScript, see the script titled JavaScript for Product Key Example.

AcroTeX PDF Blog

www.AcroTeX.net

16. Keystroke/Format of Combo Box: Editable versus Non-Editable

As with text fields, the values of `event.change`, `event.changeEx`, and `event.willCommit` are used to control what you want your Keystroke script to process.

- `event.willCommit=false`
  - When `event.change!=""`, the user has made a change, either by selecting an item from the popup list, or by typing in text. When `event.change==""` the user has backspaced, or has cleared the reset the field. Normally, we don't want to process a keystroke when `event.change==""`.
  - When `event.changeEx!=""`, the user has selected an item from the popup list; when `event.changeEx!==""`, the user is typing or pasting in text.
- `event.willCommit=true`. The user has committed the selection, then `event.value` is the face (or appearance) value of the selection (either the text the user has typed or pasted in, or one of the items included in the combo list).

When you create a Keystroke script for a combo box that sets the value of `event.change`, the script applies not only to the text the user enters into the combo input box (in the case the combo box is editable), but also to the appearance value—not the export value—of the pop-up list of the combo box. This latter point is true even when the combo box is not editable. To illustrate these points, consider the following example.

Example 16.17. Extracting Basic Info from a Combo Box. The output of these two combo boxes is written to the console window.

No Custom Text **Custom Text Allowed**

Event	**Comments**
On Focus	`event.value` and `event.target.value` are both equal to the current export value
Keystroke (before commit)	`event.value` is now the face (or appearance) value, not the export value. `event.change` and `event.changeEx` are as expected.
Keystroke (commit)	`event.value` is the face (or appearance) value, not the export value.
Validate	`event.value` is the face (or appearance) value, not the export value.
On Blur	`event.value` is now the export value.

AcroTeX PDF Blog

www.AcroTeX.net

Within the Keystroke and Validate events, `event.value` is the appearance value; when the combo box is editable, the export value and the appearance value are the same. □

Example 16.18. Creating a editable combo box with Keystroke script. Editable combo boxes are shown with a red border. Select the name of your favorite Mathematician or enter the name of your own (in the case of an editable combo box). The Keystroke script changes text the to UPPER CASE.

Name your favorite Mathematician:

Here is the same combo box, but it is no longer editable.

Name your favorite Mathematician:

There are some differences between the editable combo box, with Keystroke script that changes the value of `event.change`, and a non-editable combo box that also changes the value of `event.change`. Can you see the difference? In each of the two combo boxes above, we selected the second item in the list.

```
1  Editable Combo Box                 Non-Editable Combo Box
2  Field: combo16-16                  Field: combo16-16-1
3    current value=0                      current value=0
4    event.value=Isaac Newton             event.value=Isaac Newton
5    event.change=LEONHARD EULER          event.change=LEONHARD EULER
6    event.changeEx=1                     event.changeEx=1
7  Query of combo16-16 by button      Query of combo16-16-1 by button
8    current value=LEONHARD EULER         current value=1
9    currentValueIndices=-1               currentValueIndices=1
10                                        Appearance value=Leonhard Euler
11                                        Export value=1
```

Now select the third item in each list, Stefan Banach. More slight differences appear.

```
1  Editable Combo Box                 Non-Editable Combo Box
2  Field: combo16-16                  Field: combo16-16-1
3      current value=LEONHARD EULER         current value=1
4      event.value=LEONHARD EULER           event.value=Leonhard Euler
5      event.change=STEFAN BANACH            event.change=STEFAN BANACH
6      event.changeEx=2                     event.changeEx=2
7  Query of combo16-16 by button      Query of combo16-16-1 by button
8      current value=STEFAN BANACH           current value=2
9      currentValueIndices=-1               currentValueIndices=2
10                                          Appearance value=Stefan Banach
11                                          Export value=2
```

Observations. Internally (`event.value`, `event.change`, `event.changeEx`), there is no difference between editable and non-editable combo boxes; however, when queried externally

AcroTeX PDF Blog

www.AcroTeX.net

using the push buttons, there are differences. Recall, that *Field*.currentValueIndices returns the index of the item from the combo list that is selected, and that when the user enters text into the combo edit box, it returns a value of -1. If you compare the values of f.currentValueIndices in the first column and the second column, you see that in the first column f.currentValueIndices=-1, which means that the user has entered the text, which he did not; the column on the right reflects the correct state of the combo box. In the second set of comparisons, there is a difference in current value; the left side reports the formatted face value, while on the right, the export value is reported.

Summarizing the observations: When the combo box is editable and the Keystroke script sets event.change, Acrobat/AR treats it as if the user has entered the value. From the external view, we can't determine if the user entered the value or just selected an item from the list.

Here is the same editable combo box in which event.change is not set. The behavior is just like a non-editable combo box. When the user enters text in the combo edit field, we can detect it externally.

Name your favorite Mathematician: □

Example 16.19. **Modifying user input without modifying list items.** This example illustrates the "proper" way of writing a Keystroke script for an editable combo box; we process the keystrokes of user input, and ignore the selection of one of the items in the combo list.

To modify only the user input, leaving the items in the list unchanged, use event.changeEx to determine if the change is from the user. If event.changeEx=="", the user is typing in text.

Name your favorite Mathematician:

The Keystroke script is given below:

```
1  if (event.change!="") {
2      if (!event.willCommit && event.changeEx=="") {
3          event.change=event.change.toUpperCase();
4      }
5  }
```

Here, I've removed all the console.println lines to simplify the listing. In line (1), we process only if there is a change (this allows, back spacing by the user). In line (2), if the text is not committed, and event.changeEx=="" (which means the user is entering or typing in text), the we set event.change to upper case, line (3). □

To process only the text input by the user, the Keystroke script should have the form

AcroTeX PDF Blog

www.AcroTeX.net


```
1 if (event.change!="") {
2     if (!event.willCommit && event.changeEx=="") {
3         <process text input by user here>
4     }
5 }
```

Example 16.20. Changing event.value when event.willCommit is true. Unlike setting `event.change`, which effects the appearance of the value (even when the user simply selects an item from the list), setting the value of `event.value` when `event.willCommit` is `true` effects only the text entered by the user. Let's look at an example.

Name your favorite Mathematician:

This combo box behaves correctly relative to internally and externally examination. The Keystroke code is

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

□

While we're at it, we might as well look at the Format script.

Example 16.21. Formatting a combo box.

Name your favorite Mathematician:

Here is the same combo box, but it is no longer editable.

Name your favorite Mathematician:

The Format script is

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

□

Let's leave the Mathematicians now, I think you've learned their names by now. Those examples are my little service to you. DPS

The Keystroke script need not set `event.change` or `event.value`. An important role the Keystroke plays is to accept or reject Keystroke input. Perhaps, you want to accept only numerical values, the Keystroke script can certainly do that.

The following example uses a Keystroke script to change the combo box from non-editable to editable, for reasons explained in the example.

Example 16.22. Selecting a State from a Combo, or entering your own. Perhaps a not well-known feature of a combo box is that it search for an item in the list based on first letter. If the user presses the `A` key, when the combo box is focused, the first item that begins with the letter `A` is displayed, pressing `A` goes to the next item beginning with the letter `A`, and so on. Eventually, you cycle back to the top of the list. This feature is available only when the combo

AcroTeX PDF Blog

www.AcroTeX.net

box is *not editable*. This example offers the best of both worlds: the automatic seek feature, and the ability to enter custom text.

The combo box below lists the states of the union of the United States, as well as its various protectorates. The last item in the list is `Other`. The combo box starts out non-editable—try pressing the other `O` several times; eventually, you'll get to `Other`, press enter to select `Other`, the border of the combo box turns red to indicate the combo box is now editable. Enter some text, press enter to commit the text, the border turns black again, and the combo box is now non-editable and seeking is restored. Cool!

Select a state:

The Keystroke script is

```
if (event.willCommit) {
    if (event.value.replace(/\s*/g,"")=="") event.value="Please Select";
    if (event.value=="Other...") {
        event.target.editable=true;
        app.alert("Enter another state or protectorate of the US not listed.");
        event.target.strokeColor=color.red;
    } else {
        event.target.editable=false;
        event.target.strokeColor=color.black;
    }
}
```

Example 16.23. Formatting a Combo box using built-in functions.

Rate the AcroTEX PDF Blog in terms of technical information delivered, and value to your own work. Select a rating, 0 is a low rating, and 10 is the best. Feel free to enter your own rating, negative ratings will be rejected.

Rate the AcroTEX PDF Blog:

Here, the Keystroke script is

```
AFNumber_Keystroke(1,0,1,0,"",true);
if (event.willCommit && event.rc ) {
    if (event.value > 10 ) var cMsg="<funny saying>";
    if ( event.value > 7 && event.value <= 10 )
        var cMsg="<humorous quip>";
    if ( event.value > 4 && event.value <= 7 )
        var cMsg="<clever quote>";
    if ( event.value <=4 ) var cMsg="<sad saying>";
    app.alert(cMsg);
}
```

AcroTeX PDF Blog

and the Format script is

```
AFNumber_Format(1,0,1,0,"",true);
```

17. Selection Change for the List Box

This is page 29, that's a lot of pages! I still have a section to go. I'll make it short.

Under the Selection Change tab of the List Box Properties dialog box, you can access the editor for entering script. Acrobat implements the Selection Change script as a Keystroke script, so all the comments and examples of Keystroke scripts already presented apply; in particular, the comments on the combo are probably still valid. Of course, the List Box does not allow the use to enter custom text, that's good!

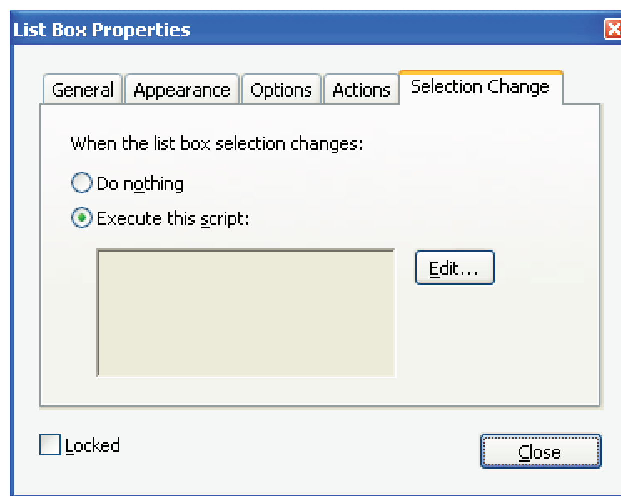

Figure 7: Selection Change Tab of the List Box

Example 17.24. Extracting Info from a List Box. We extract basic information from a list box under in two cases, when the option Commit Selected Value Immediately is check and cleared, this option is found in the Option tab of the List Box Properties dialog box. The output of these list boxes are written to the console window, so you must Open your console.

No Immediate Commit **Immediate Commit**

www.AcroTeX.net
AcroTeX PDF Blog

www.AcroTeX.net

When an item is selected from the list box on the left, the "Selected" script is executed, while for an item selected from the list on the right, the "Selected" script is executed followed immediately by the "Committed" script. Note the `event.value` is the export value of the field (this is true for the Keystroke, Focus, and Blur events), as is `event.changeEx`, while `event.change` reflects the face (or appearance) value of the field, during the Keystroke event before the data is committed.

Based on the results given in the console window, it is easy to write script that takes action based on the selected value, whether it be before the date is committed, or not. □

The other interesting feature of the list box is its ability to select multiple values from the list. Recall that in AcroTEX Blog #17 we surveyed external methods of extracting information from a list box, including multiple selection. Now, we look at internal methods.

Example 17.25. **Extracting Info from a List Box: Multiple Selection** We extract basic information from a list box when the list box allows multiple selection. (Immediate commit is not available here.) The output of these list boxes are written to the console window, so you must Open your console.

Multiple Selection

You'll note that internally (through `event.change`, `event.changeEx`) cannot determine if the user has made multiple selections. The properties `event.change`, `event.changeEx` only reflect the most recent selection by the user. The property *Field*. `currentValueIndices` is of no use in the here, as its value(s) reflect the state of the field when the field brought into focus.

Knowledge of what the user finally selected can first be detected in the Blur event, as is seen from the console output. When there were multiple selections, you can get the multiple selection from during the Blur event with `event.target.value`, this is an array of export values the user selected; or it may just a string, in this case, it is the export value of the single item selected from the list. □

Example 17.26. **Comparing the Combo and List Boxes.** Both of these form elements are choice fields. Combo box can have custom text (list box cannot); list box can have multiple selection (combo box cannot). There are slight differences in event.value between these two boxes that should be pointed out.

The list box does not have immediate commit option selected, and the combo box does not allow custom text. The output the console window only reports the Keystroke script.

AcroTeX PDF Blog

www.AcroTeX.net

List Box **Combo Box**

The only difference between these two fields is in the value of `event.value`: For the list box it is the export value; for the combo box it it the appearance value. Because the combo box can have custom, the event.value must necessarily reflect what the user has entered, so the face or appearance value is used for the combo; no custom text is allowed for the list box, so the export value is used. □

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

AcroTeX PDF Blog