



AcroTeX.Net

AcroTeX PDF Blog

Processing Acrobat Forms using JavaScript

External Processing of a Field

Part 6: Choice Fields, the List Box

D. P. Story

Table of Contents

9 Choice Fields: The List Box	3
9.1 Getting and Setting the Value of a List Box	3
9.2 Getting the Number of Items in a List	3
9.3 Getting and Setting	3
• Using <i>Field.value</i>	4
• Using <i>Field.currentValueIndices</i>	5
• Using <i>Field.getItemAt</i>	6
9.4 Creating, Inserting, Deleting, and Clearing a List	7
9.5 Changing the Appearance and the Properties of a List Box	9
9.6 The General Tab	9
9.7 The Appearance Tab	10
9.8 The Options Tab	11

9. Choice Fields: The List Box

Ahh, aPDFBlog #17, you may not know this, but 17 is my *formerly favorite number*.¹ Now, on to list boxes.

A *list box* is a rectangular region what contains a scrollable list of several text items, one or more of which may be selected as the field value. A vertical scroll bar is available when the height of the list box is not large enough to display all items in the list. The list box to the left is a list some a small subset of \LaTeX packages that I've written. We'll use this list box to illustrate some of the JavaScript methods and properties for extracting information about the box, and which items are selected.

9.1. Getting and Setting the Value of a List Box

Each item in a list box has an export value and an appearance (or face) value.

- The *export value* of an item in a list box is not seen by the user. It is the value, or values, of the selected item(s) that are submitted to a server-side script, for example.
- The *appearance (or face) value* of an item is the text string you see in the list. This is also referred to in various parts of *JavaScript for Acrobat API Reference* as the *user value*.

In addition to these two, there is an *index value* that can be used to get and set the items in a list. The export value and face value of a list item are internally stored in an array (the *options array*). The index value corresponds to the items position in the array. An index value of 0 is the first item in the list, an index value of 1 is the second item in the list, and so on.

9.2. Getting the Number of Items in a List

The number of items in a list box can be obtained from the `Field.numItems` property. For example, the number of items in the list box above is obtained by pushing button below.

```
1 var f=this.getField("myList");
2 app.alert("There are "+f.numItems+" items in this list.");
```

9.3. Getting and Setting

We can extract information from a list using various Field properties and methods:

- `Field.value`: get and set by export value;
- `Field.currentValueIndices`: get and set by index value;
- `Field.getItemAt`: get either export value or appearance value by index value.

¹I have another formerly favorite number, it will a awhile before we get to that number.

We take up each of these in turn.

- Using *Field.value*

The *Field.value* property can be used to get and set the value(s) of a list by using the export value.

Example 9.1. We get the value of a list box. Select an item in the list, and press the Get button.

```

1 var f = this.getField("myList9-1");
2 var strMsg="The export value selected is \""+f.value+"\".";
3 app.alert(strMsg);

```



A list box can also allow multiple selections. The user can Ctrl+Click or Shift+Click to select multiple items. *Field.value* returns a string, the export value, if a single item is selected, and an array of export values, otherwise.

Example 9.2. We get the value of a list box with multiple select. Select one or more items in the list, and press the Get button.

```

1 var f = this.getField("myList9-2");
2 var strMsg="The export value selected is "
3   + (( typeof f.value == "string" )
4     ? "a single value, \""+f.value+"\"."
5     : "an array of values, "+ f.value.toSource()+".")
6   + " You selected "+f.value.length+" items from the list.");
7 app.alert(strMsg);

```

Comments. *f.value* returns a string, if a single item is selected, or an array, if multiple values are selected. In building the message in line (2)–(6), we test whether *f.value* is a string in line (3), and modify our message accordingly.

The Set button script is

```

1 var f = this.getField("myList9-2");
2 f.value=["AeB", "APB"];

```

Setting dirties the document, save rights are needed for Adobe Reader to make any change

permanent. □

• Using `Field.currentValueIndices`

The `Field.currentValueIndices` property works with the indices of the array of list items. It gets either a number (the 0-based index of the selected item) or an array of numbers (if multiple selection is allowed, and multiple items were selected).

Similarly, to set a single item, set this property to an integer value (0-based), To select multiple items set this property to an array of integers (if multiple selection is permitted). *JavaScript for Acrobat API Reference* states that this is the preferred method of setting the values of a list box (and combo box).

In Adobe Reader, we can get and set the items; however, for setting, we need additional Forms usage right to save the changes.²

Example 9.3. Get and set in multiple selection list box. Select one or more items in the list, and press the Get and Set buttons.

```

1  var f = this.getField("myList9-3");
2  if (typeof f.currentValueIndices=="object") {
3      var l=f.currentValueIndices.length;
4      var itemStr=""
5      for (var i=0;i<l-1; i++)
6          itemStr+=(aNumberf.currentValueIndicesi+", ");
7      itemStr+=("and "+aNumberf.currentValueIndicesl-1);
8  }
9  var strMsg="You selected "
10     + (( typeof f.currentValueIndices == "number" )
11         ? "the "+aNumberf.currentValueIndices+" item."
12         : f.currentValueIndices.length+" items, the "+ itemStr+"."
13         + " Am I right? dps");
14  app.alert(strMsg);

```

Comments. We test `f.currentValueIndices` several ways, in line (2), we see if the return value is an array, in line (10), we test if the return value is a number.

²LiveCycle Reader Extensions may be used to activate additional functionality within Adobe Reader. Form usage rights will allow for a document full-save.

Here is the script for the Set button:

```
1 var f = this.getField("myList9-3");
2 f.currentValueIndices=[0,2,5];
```



`Field.currentValueIndices` is Acrobat's preferred way of getting and setting the value of a list box (or combo box).³ One reason for this is that `Field.currentValueIndices` is a general property that does not require knowledge of the face or export value of the list (or combo) box. It is ideal, therefore, for use in a general script for managing list boxes or combo boxes. For this reason, the use of `Field.getItemAt` is also recommended, see the next section.

• Using `Field.getItemAt`

The `Field.getItemAt` method may be viewed as a companion method to the `Field` property `Field.currentValueIndices`, or not.

`Field.currentValueIndices` can be used to get the indexes of the currently selected items, but does not return the export or face values. `Field.getItemAt` takes an index as one of its parameters, and returns either the export value or the appearance value, depending on the value of the second parameter.

```
Field.getItemAt({ nIndex: Integer, bExportValue: Boolean })
```

If the `bExportParameter` is set to true, this method returns the export value, otherwise it returns the appearance value.

Example 9.4. Get and set using `Field.getItemAt`. Select one or more items in the list, and press the Get button.

```
1 var f = this.getField("myList9-4"), strMsg="You selected:\r";
2 if (typeof f.currentValueIndices=="object") {
3     var l=f.currentValueIndices.length;
4     for (var i=0;i<l; i++)
5         strMsg+=(" \u2022 "+f.getItemAt(f.currentValueIndices[i], false)
6             +" (" +f.getItemAt(f.currentValueIndices[i], true)+")\r");
7 } else
8     strMsg+=(" \u2022 "+f.getItemAt(f.currentValueIndices, false)
9         +" (" +f.getItemAt(f.currentValueIndices, true)+")\r");
10 strMsg+="\rOver and out. dps";
11 app.alert(strMsg);
```



³There is one exception, that will be covered when we discuss the combo box in the next blog.

The use of `Field.currentValueIndices` and `Field.getItemAt` is recommended for writing general purpose scripts that manage list boxes or combo boxes. However, `Field.getItemAt` is a general method for snooping into the list of a list box (or combo box) and need not be used with `Field.currentValueIndices`.

9.4. Creating, Inserting, Deleting, and Clearing a List

There are several `Field` methods for managing the items in a list box. These are...

- `Field.setItems` for creating a new list of items for a list;
- `Field.clearItems` for clearing away the old items from a list;
- `Field.insertItemAt` for inserting additional items into an existent list;
- `Field.deleteItemAt` for deleting items from a list.

These methods may work for Adobe Reader, but any changes are not permanent, unless the document has a full-save usage Reader rights.

Example 9.5. This example illustrates `Field.setItems` and `Field.clearItems`. Select either the AcroTeX and Acrobat radio buttons, then select, at your option, any of the items in the list box.

AcroTeX Acrobat

Below left is the `MouseUp` action of the AcroTeX radio button, the code for the Acrobat button is displayed below right.

```

1  var f = this.getField("myList9-5");
2  var aList1=[
3      ["AcroTeX eDucation Bundle","AeB"],
4      ["AeB Pro","AeBPro"],
5      ["AcroTeX Presentation Bundle","APB"],
6      ["@EASE","ATB"],
7      ["JavaScript Jeopardy Game","JJG"],
8      ["Das Puzzle Spiel","DPS"]
9  ];
10 f.setItems(aList1);

```

```

1  var f = this.getField("myList9-5");
2  var aList2=[
3      "Acrobat Professional Extended",
4      "Acrobat Professional",
5      "Acrobat Standard",
6      "Adobe Reader"
7  ];
8  f.setItems(aList2);

```

Comments: The two buttons show the two formats of the elements of the array that can be fed to the `Field.setItems`. In the format on the left, each element is a two-element array, the first element is the face value (appearance value) and the second is the export value. For the format of the array on the right, each element is a string. The value of the string is used as both

the appearance and export value of the item. The two element formats can be mixed together, though, they are not here. Finally, after we have built our list, we call `f.setItems(aList1)` or `f.setItems(aList2)`.

The Reset push button code is

```
1 var f = this.getField("myList9-5");
2 f.clearItems();
3 this.resetForm(["rb9-5", "txt9-5", "myList9-5"]);
```

Comments: We clear the list (1)–(2), then reset the other fields (3);

Finally, though we don't discuss this code here, take a look at the Section Change script of the list box. In this current thread of blogs, we are examining the *external processing* of a field. No doubt, we'll examine this code in detail in the blogs on *internal processing* of a field. □

To modify an existent list—inserting an item or deleting an item—use `Field.insertItemAt` and `Field.deleteItemAt`.

`Field.insertItemAt` takes three parameters:

```
Field.insertItemAt({ cName: String, cExport: String, nIdx: Integer })
```

The `cName` parameter is the appearance value of the item, `cExport` is the export value of the item, and `nIdx` is the index at which to insert the item. If `nIdx` is 0 (the default), the item is inserted at the top of the list, and if `nIdx` is -1, the item is listed at the bottom of the list.

The `Field.deleteItemAt` takes a single parameter, `nIdx`, the index of the item to be deleted. If the item deleted is the currently selected item, make a new selection using `Field.currentValueIndices`.

Example 9.6. Insert and delete AcroTeX PDF Blog into the list below. Push the button to make this happen.

The code for the Insert/Delete button appears on the left, while the code for the Reset List button appears on the right.

```
1 var f = findAndDelete("myList9-6", "aPDFBlog");
2 f.insertItemAt({
3   cName: "AcroTeX PDF Blog",
4   cExport: "aPDFBlog",
5   nIdx: insertionPoint
6 });
7 insertionPoint = ++insertionPoint % f.numItems;
```

```
1 var f = findAndDelete(
2   "myList9-6", "aPDFBlog");
3 f.currentValueIndices=0;
```

Both sets of script call a function `findAndDelete()`, defined at the document level. This function finds and deletes the item with export value of "aPDFBlog", if it exists, and returns the field object of the list box.

Comments for the Insert/Delete button: In line (1), `findAndDelete()` is used to locate and delete the item with export value of "aPDFBlog". We then insert a new item in the list (lines (2)–(6)) at the `insertionPoint`, a global variable set up at the document level. We increment `insertionPoint`, then perform modular arithmetic, to cycle back to 0, if needed, in line (7). □

9.5. Changing the Appearance and the Properties of a List Box

As we have done in the previous blogs, we'll survey the General, Appearance, and Option tabs. Saving the other tabs (Actions, Selection Change) for another day.

9.6. The General Tab

Below, see [Figure 1](#), is a screen shot of the General tab of the List Box Properties, parallel to the dialog box, are the JavaScript properties that correspond to the elements on the General tab of the List Box Properties dialog box. The properties of the General tab, are the same as those of the other fields.

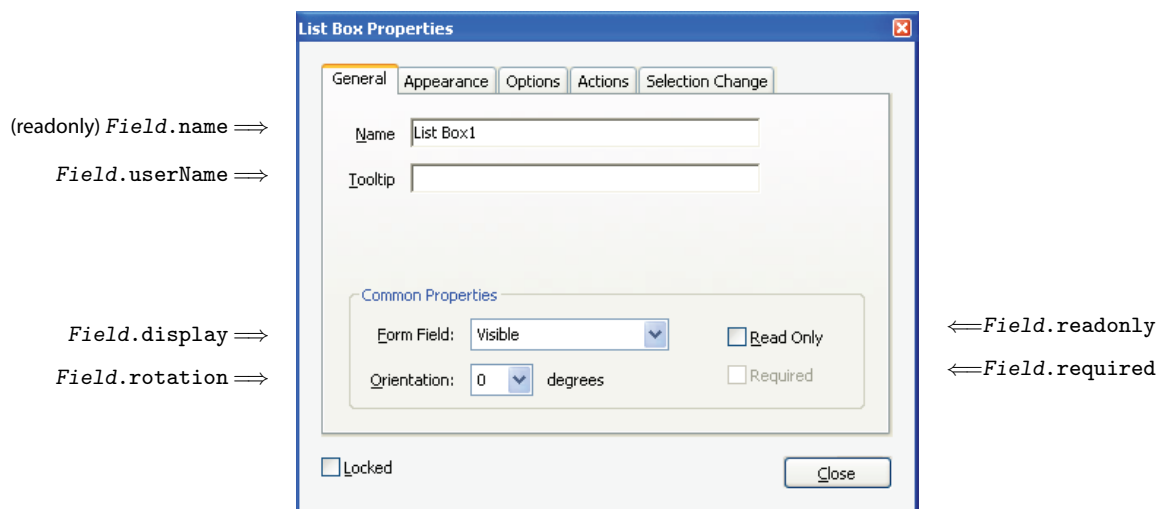


Figure 1: The General Tab of List Box Properties

We have pretty well illustrated these properties in the previous blogs, [AcroTeX PDF Blogs #13](#), [#14](#), [#15](#), and [#15](#), so no examples will be presented here. Move on!

9.7. The Appearance Tab

Let's do the same now for the appearance tab, see [Figure 2](#) on page 10.

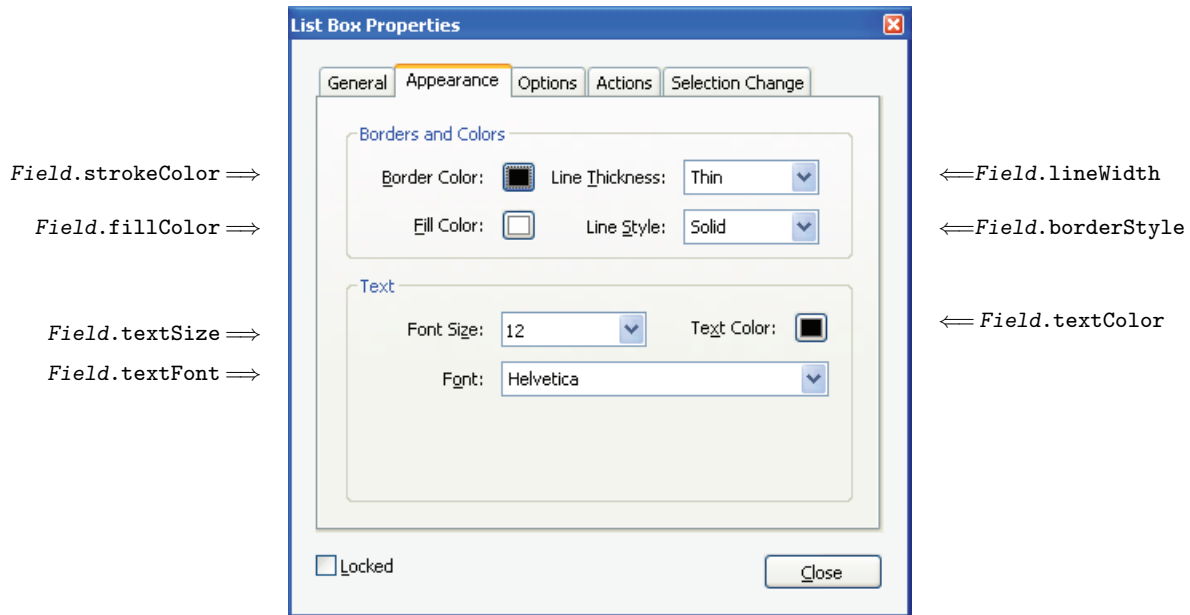


Figure 2: The Appearance Tab of List Box Properties

As with the General tab, use these properties, listed in [Figure 2](#), by first acquiring the Field object of the target field, and assigning values to the JavaScript properties. In general, some properties and methods are only available in Acrobat, and not available on Adobe Reader. See the *JavaScript for Acrobat API Reference* for full documentation, pay attention to the quick keys that accompany the properties and methods in the *JavaScript for Acrobat API Reference*.

Border Color: Determines the color of the border; *Field.strokeColor* is its counter-part in JavaScript.

Fill Color: The background color of the list box.

Font Size: The size of the text to appear in the list. Setting the *Field.textSize=0* is equivalent to selecting Auto from the drop-down list in the user interface.

Font: The font to be used to render the text items in the list.

Thickness: Use *Field.lineWidth* to set the width of the border surrounding the bounding rectangle of the list box. Possible values are 0 (no boundary), 1 (Thin), 2 (Medium), 3 (Thick).

Line Styles: How the border is rendered, the user interface offers the choices: Solid, Dashed, Beveled, Inset, and Underlined. The corresponding values for *Field.borderStyle* are

`border.s`, `border.d`, `border.b`, `border.i`, and `border.u`.

Executing `Field.borderStyle=border.s` sets the Line Style to Solid.

Text Color: This property determines the color of the the text. To set the color of the text to red, execute `Field.textColor=color.red`

We have pretty well illustrated these properties in the previous blogs, AcroTeX PDF Blogs #13, #14, #15, and #16, so no examples will be presented here.

9.8. The Options Tab

In Figure 3 we set up the correspondence between each user interface element and its JavaScript counterpart.

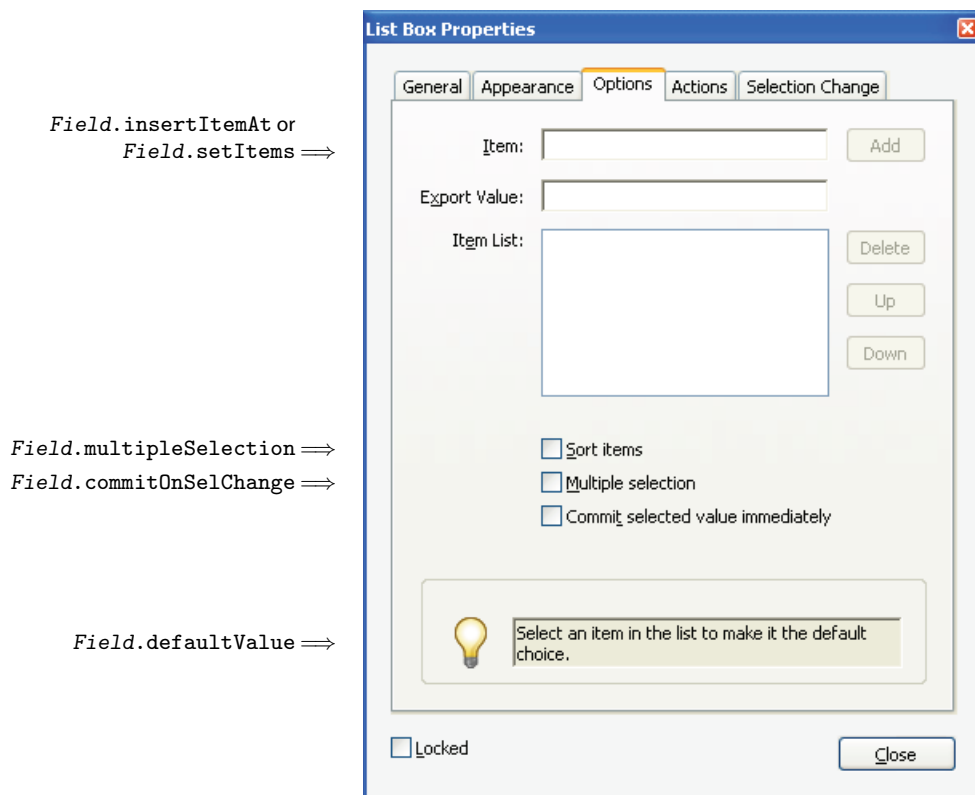


Figure 3: The Options Tab of List Box Properties

Item/Export Value/Item List: These user interface elements can be filled in by using either `Field.getItemAt`, for one item at a time, or `Field.setItems`, for all items at one time. See Examples 9.5 and 9.6.

Sort Items: There is no JavaScript property for checking this box; however, if you want the items in a list sorted, you can sort them using JavaScript. The following script should do the job.

```
1 var f=this.getField("lbName");
2 aSorted=new Array();
3 for (var i=0; i<f.numItems; i++)
4     aSorted.push(f.getItemAt(i,false),f.getItemAt(i,true))
5 aSorted=aSorted.sort();
6 f.setItems(aSorted);
```

The above code works, but I was a little nervous about it. Originally, I thought that I had to have a sorting function, like the following one.

```
var sortIt = function(a,b){
    if (a[0] < b[0]) return -1;
    if (a[0] > b[0]) return 1;
    if (a[0] == b[0]) return 0;
}
aSorted=aSorted.sort(sortIt);
```

Insert these lines in place of lines (5).

Multiple selection: Put `Field.multipleSelection=true` to allow the user to make multiple selections.

Commit selected value immediately: If `Field.commitOnSelChange=true` commits the selection immediately, even though the field has not lost focus. Losing focus is the usual way of committing the choice. The following two list boxes demonstrate the difference between the two modes.

Example 9.7. This example writes to the JavaScript Debugger Console window. Open the console window, if you are using the puny Adobe Reader, click on [console window](#). In the list box, select one, then another item in the list box (no multiple selections allowed), nothing is written to the window. Now click you mouse outside the list box, the data is committed, and a message is written to the window.

Now click on the push button, the caption should say **commOnSelChange**. Repeat the exercise above. As you make selections within the list box, they are reported to the console window. You don't have to commit the selection by clicking outside the list box.



Select an item in the list to make it the default choice. Setting `Field.defaultValue` to the *export value* of one of the items makes that item the default item. The default value is the item that is highlighted when the list box is reset, perhaps using `this.resetForm()`. Acrobat is needed to save any changes and to make them permanent. For example, click the button then go to the page containing the "myList" list box, on [page 3](#). If luck is with us, the "AeB Pro" item is highlighted (and is now the default item). The code for the button is...

```
1 var f = this.getField("myList");
2 f.defaultValue="AeBPro";
3 this.resetForm("myList");
```

We could have used `f.getItemAt(1,true)` to retrieve the export value of the second item in the list, which is the position of "AeB Pro", and made this assignment

```
\texttt{f.defaultValue=f.getItemAt(1,true)}
```

in line (2).

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