

**Creating Popup Menus  
with the popupmenu Package**

**D. P. Story**

The L<sup>A</sup>T<sub>E</sub>X source file for this [AeB Blog](#) is attached to this document. [Click here](#) to open the attachments panel. The attachments include the source file for this blog, and a zip file containing the popupmenu package, with an example file. The package is contained in the file `popupmenu.zip.txt`, save this file as `popupmenu.zip`, unzip in a file on your latex search path, `latex popupmenu.ins` to unpack the distribution to create `popupmenu.sty`. Unless you use distiller to create PDF, you cannot successfully compile this blog file, but you can copy and paste the examples.

## 1. Introduction

I've recently written a short package to create popup menus; this package uses the `app.popupMenuEx` method of Acrobat JavaScript. The method takes as its argument a menu structure. This menu structure can be conveniently defined by a series of  $\LaTeX$  environments.

The package makes no special requirements, and can be used by users of `pdftex`, `dvipdfm`, and `dvips` (and my own favorite, `dvipsone`).

## 2. First Example

Let's begin with a very simple example. We'll create a menu with three items in it. We use the `popupmenu` environment like so,

```
\begin{popupmenu}{myFirstMenu}
  \item{title=Aeb is Good}
  \item{title=AeB Pro is Great}
  \item{title=AcroTeX rocks!}
\end{popupmenu}
```

This code does two things: It creates a command `\myFirstMenu` that contains the menu structure of the menu just defined, this menu structure—expressed as a JavaScript array—is assigned to a JavaScript variable `myFirstMenu`.

The JavaScript array containing the menu structure can be placed as document level JavaScript, or as field level JavaScript. Each of these will be illustrated.

## 2.1. Document level JavaScript

The popupmenu named `myFirstMenu` is placed in the preamble of this document and inserted into the document JavaScript. The preamble for this document has the following lines

```
\begin{popupmenu}{myFirstMenu}
  \item{title=Aeb is Good}
  \item{title=AeB Pro is Great}
  \item{title=AcroTeX rocks!}
\end{popupmenu}
\begin{insDLJS}[myFirstMenu]{mymenus}{Menu Data}
\myFirstMenu
\end{insDLJS}
```

Here, we use the `eforms` package to place this code at the document level.

Now that the menu array is in place, we can use it using the `\popUpMenu` command, defined in the `popupmenu` package.

The code for this button is

```
\pushButton[\CA{AeB}\S{S}\AA{\AAMouseEnter{\JS{%
var cChoice = \popUpMenu(myFirstMenu);\r
if ( cChoice != null )
  app.alert("You chose the \""+cChoice+"\" menu item");
}}]{mymenu1}{-}{11bp}
```

We implement this menu as a mouse over event. When you enter the button face, the menu will “pop up.” The return value is the menu name (the value of the `title` key) unless otherwise specified.

## 2.2. Field level JavaScript

It is an advantage to place the menu array at the document level if the menu is used several times throughout the document. If the menu is only used once, there is no harm to place it as field level JavaScript, here is an example of that.

Here an example of a field level menu structure.

The code for this button is

```
\begin{popupmenu}{myFieldLevelMenu}
  \item{title=AcroTeX rocks!}
  \item{title=Aeb is Good}
  \item{title=AeB Pro is Great}
\end{popupmenu}
```

Here an example of a field level menu structure.

```
\pushButton[\CA{AeB}\S{S}\AA{\AAMouseEnter{\JS{%
\myFieldLevelMenu\r
var cChoice = \popupMenu(myFieldLevelMenu);\r
if ( cChoice != null )
  app.alert("You chose the \""+cChoice+"\" menu item");
}}]{mymenu2}{\11bp}
```

In the mouse over JavaScript code you'll notice that I've inserted the command `\myFieldLevelMenu`, which is the name of the new menu. I've rearranged the order of the menu items to assure you that menu is different from the previous one.

## 3. Specifying a return value

The `app.popupMenuEx` method returns the name of the item, unless a return value is explicitly defined. To specify a return value, use the return key. Below

is an example, we return a URL, and the action is to launch this URL.

We placed the following code in the preamble, and inserted it into the document JavaScript.

```
\urlPath{\aebhome}{http://www.math.uakron.edu/~dpstory}
\begin{popupmenu}{AeBMenu}
  \item{title=AeB, return=\aebhome/webeq.html}
  \item{title=-}
  \item{title=AeB Pro,return=\aebhome/aeb_pro.html}
  \item{title=Graphicxsp,return=\aebhome/graphicxsp.html}
  \item{title=eqExam,return=\aebhome/eqexam.html}
\end{popupmenu}
```

Note the use of the return key, we provide URLs. I've also included a special menu item with a title of `title=-`, when Acrobat (AR) sees that a hyphen is the title, it puts in a separator bar, as seen in the example that follows. Also, in the preamble, we use a command `\urlPath` to define the URL, and to save it under the name `\aebhome`.

We now create a button:

The code for the button is

```
\pushButton[\CA{Packages}\AA{\AAMouseEnter{\JS{%
var cChoice = \popUpMenu(AeBMenu);\r
if ( cChoice != null ) try {app.launchURL(cChoice);}catch(e){}
}}]{aebmenu}{}{11bp}
```

The return value is an URL string, and we use `app.launchURL` to go to that address.

## 4. Marking and Enabling a Menu Item

A menu item can be marked (with a check mark) or disabled; the keys `marked` and `enabled` are used for that purpose; `marked` has a value of `false` by default, and `enabled` has a value of `true` by default. For example, the code

```
\item{title=D. P. Story, enabled=false, marked}
```

marks the menu item with a check, and disables the item.

In the next example, we try to dynamically mark the menu items. We'll use the `mySecondMenu`:

The menu structure of `mySecondMenu` is the same as `myFirstMenu` with the exception of the return value. The first menu items are contrasted below:

```
// from myFirstMenu
  \item{title=Aeb is Good}
// from mySecondMenu
  \item{title=Aeb is Good,
        return={[\itemindex,'AeB is Good']}}
```

The return value for `mySecondMenu` is a string (the return value is supposed to be a string) that contains the index into the array of the menu item and the title of the menu item. The command `\itemindex` is calculated from within the `popupmenu` environment, and is available as a (part of the) return value. The document level function `processMenu()` uses the return value to turn the `marked` property on and off. If a marked menu item is chosen, that menu item is unchecked.

## 5. Creating Submenus

You can create a popup menu with submenus by using the submenu environment. The submenu environment takes one argument, the same key-value pairs that are used for the `\item` command. A title key is required, there is no return value, it can be marked, but not dis-enabled.

We illustrate by example.

```
\begin{popupmenu}{AeBMenuSubMenus}
  \item{title=AeB, return=\aebhome/webeq.html}
  \item{title=eqExam,return=\aebhome/eqexam.html}
  \item{title=-}
  \begin{submenu}{title=AeB Pro Family}
    \item{title=Home page,return=\aebhome/aeb_pro.html}
    ...
    \item{title=AcroFleX,return=\aebhome/acroflex.html}
  \end{submenu}
  \item{title=-}
  \begin{submenu}{title=Web Sites}
    \item{title=AcroTeX,return=\aebhome/acrotex.html}
    ...
    \item{title=Home Page of D. P. Story,return=\aebhome}
  \end{submenu}
\end{popupmenu}
```

This menu is defined in the preamble of this document, and inserted at the document level.

The code for the push button and the text field are given below. The button action gets the choice from the menu and populates the text field to the right. The JavaScript for the text field includes a mouse up action for launching the URL. A tooltip is also defined for the text field.

```

\pushButton[\CA{AeB Web Addresses}\S{S}
\AA{\AAMouseEnter{\JS{%
var cChoice = \popUpMenu(AeBMenuSubMenus);\r
if ( cChoice != null ) {\r\t
    var f=this.getField("txtAeBURLs");\r\t
    f.value=cChoice;\r
}}}}]{myaeburls}{-}{11bp}\quad
\textField[\TU{Click to go to this URL}\A{\JS{%
if (event.target.value != "")
    try {app.launchURL(event.target.value);}catch(e){}
}]}{txtAeBURLs}{3in}{11bp}

```

Here is the same example, but the text field is a rich text field, and the button populates it with rich text, the URL is now blue and underlined, make it look more like a link.

The JavaScript for the button action for processing the popup menu is

```

var cChoice = \popUpMenu(AeBMenuSubMenus);
if ( cChoice != null ) {
    var f=this.getField("rtxtAeBURLs");
    var spans=new Array(); spans[0]=new Object();

```

```
spans[0].textColor=color.blue;  
spans[0].underline=true;  
spans[0].text=cChoice;  
f.richValue=spans;  
}
```

## 6. List of properties

We finish this article with a listing the key-values recognized by `\item` and the argument of the `submenu` environment:

- `title`: The menu item name, which is the string to appear on the menu item. The value of "-" is reserved to draw a separator line in the menu. The JavaScript key is `cName`.
- `marked` (optional): A Boolean value specifying whether the item is to be marked with a check. The default is `false` (not marked). The JavaScript key is `bMarked`.
- `enabled` (optional): A Boolean value specifying whether the item is to appear enabled or grayed out. The default is `true` (enabled). The JavaScript key is `bEnabled`.
- `return` (optional): A string to be returned when the menu item is selected. The default is the value of `title`. The JavaScript key is `cReturn`.

The `title` key is required, the others are optional. The `enabled` and `return` keys are ignored (by Acrobat/Adobe Reader) when passed as an argument of `submenu`.

The JavaScript property, `oSubMenu`, of the menu structure passed to the method `app.popUpMenuEx` has no  $\LaTeX$  counterpart. This property key-value pair is automatically inserted by the `submenu` environment.

That's all for now, I must get back to my retirement!