

AcroTeX.Net


The manual for the popupmenu package

D. P. Story

Table of Contents

1	Introduction	3
1.1	Sample files	3
1.2	Options, requirements, and workflows	3
2	The <code>popupmenu</code> environment	3
3	The <code>popupmenu</code> environment	4
4	Executing a pop-up menu	5
4.1	Declaring <code>popupmenu</code> in the body	7
4.2	Declaring <code>popupmenu</code> in the preamble	7
5	Remarks on <code>ps2pdf</code>	9



1. Introduction

Acrobat JavaScript has an interesting function, `app.popUpMenuEx()`, that we exploit in this package. The function takes as its argument an array of structured menu items and displays these items as a pop-up menu. When an item is selected, a value is returned, which can then be acted upon in some way. Here is a simple example, pass your mouse pointer over the button: . This documentation describes the environments, commands, and JavaScript required to create such pop-up menus.

1.1. Sample files

The file `pu-exmpls.tex` is the only demo file; it incorporates the examples of this documentation, as well as few other bits and pieces.

1.2. Options, requirements, and workflows

tracking **Options.** There are two options for this package `tracking` and `!tracking` (the default). The push button  introduced earlier is an example of a menu system with *no tracking* (`!tracking`). As you selected menu items appearance of the item does not change. When the `tracking` option is taken, the menu keeps track of which menu item is selected by placing a check mark to the left of the menu item; for example, , notice as you select items, a check mark appears; select another item, the previous check mark is removed, and the latest item selected is now checked. More on tracking in [Section 4.2](#).

Requirements. This package requires the `eforms` package, which is part of `acrotex`.¹

Workflows. This is a general \LaTeX package, any workflows can be used to build a `popupmenu` document: `pdflatex, lualatex, xelatex, or dvips -> (distiller | ps2pdf)`.²

2. The `popupmenu` environment

To generate a pop-up menu using `app.popUpMenuEx()` you need to pass to it through its argument a menu-array. The most convenient way of creating this “menu-array” is with the `popupmenu` environment. Before discussing the full syntax of `popupmenu` we reproduce the `popupmenu` environment that produced the `Intro` button in first paragraph of this section:

```
\begin{popupmenu}{Intro}
  \item{title=title,return=Title: The manual for the popupmenu package}
  \item{title=author,return=Author: D. P. Story}
  \item{title=package name,return=Package: popupmenu}
\end{popupmenu}
```

This environment defines a JavaScript variable `Intro` and a \LaTeX command `\Intro` that expands to what you see below.

¹<http://www.ctan/pkg/acrotex>

²`popupmenu` uses document JavaScript when `\puUseMenus` is expanded in the preamble; `ps2pdf` does not support the creation of document JavaScript. If all menu structures are defined as field scripts, `ps2pdf` should work. See Section 5 for more information.

```
var Intro = [
  {cName: "title", cReturn: "Title: The manual for the popupmenu package"},
  {cName: "author", cReturn: "Author: D. P. Story"},
  {cName: "package name", cReturn: "Package: popupmenu"}
];
```

The above is a properly formed “menu-array”.

The rollover push button was created with the following code:

```
\pushButton[\CA{Info}\BC{} \S{S}\H{N}\AAmouseenter{%
  var cChoice = \popUpMenu(Intro);\r
  if ( cChoice != null ) app.alert(cChoice); }]{intro}{11bp}
```

The mouse-enter JavaScript is `var cChoice = \popUpMenu(Intro)`. The array name (Intro) is passed to the convenience command `\popUpMenu` to get the return value `cChoice`.³ Finally, if nonempty, an alert dialog box is emitted with the return value displayed in it. BTY,⁴ `\popUpMenu` expands to the JavaScript method `app.popUpMenuEx()`.

3. The popupmenu environment

This general syntax for the popupmenu environment is,

```
\begin{popupmenu}{<name>} ❶
\item{title=<str>,marked=<true|false>,enabled=<true|false>,return=<str>} ❷
\item{title=-} ❸
...
\begin{submenu}{title=<str>,marked=<true|false>} ❹
\item{title=<str>,marked=<true|false>,enabled=<true|false>,return=<str>} (1)
\item{title=-}
...
\end{submenu}
...
\end{popupmenu}
```

At the top-level ❶, the popupmenu environment takes one argument. The `<name>` argument plays two roles: (1) it becomes the name of the JavaScript *menu-array*; (2) it becomes a command name `\<name>` that expands to the JavaScript menu-array. For this reason, `<name>` must consist of ASCII letters only. The body of popupmenu consists of one or more `\item` commands (❷ and ❸). The body may contain zero or more submenu environments (❹); submenu may then contain one or more `\item` commands. You can have sub-menus inside other sub-menus.

Generally, some underlying JavaScript, such as when the tracing option is in force, sets a menu item to `marked=true` or `marked=false`; so as a rule, it is not recommended to initially specify the marked key.

³The name of the return variable is your choice, you can say `var retn=\popUpMenu(Intro)`, for example.

⁴By the way

Discussion of `\item{<KV-pairs>}`. There are four key-value pairs.

`title=<str|->` (required) The value of `title` (`<str>`) is the menu item title; a value of `'-'` is reserved to signal that a separator line should be drawn.

`marked=<true|false>` (optional) If `true`, the menu item is marked with a check. The default is `false` (not marked). Leave this key to JavaScript.

`enabled=<true|false>` (optional) If `true` the item is to appear enabled; otherwise the menu item is grayed out. The default is `true` (enabled).

`return=<str>` (optional) A string to be returned when the menu item is selected. If `return` is not specified or has no value, the value of the `title` key is returned. There is a special return value; if `<str>` is the word `'none'`, the return value will be `null` (no action). In this case, the value of the `title` key can be used as a heading. (There is an example of the `'none'` value in Section 4.)

Discussion of `submenu{<KV-pairs>}`. The argument of `submenu` takes only two key-value pairs: `title` and `marked`, see descriptions above.

Placement of the `popupmenu` environment. The environment may appear in the preamble or in the body of the document.

- **In the preamble.** When one or more `popupmenu` environments are declared in the preamble, their corresponding menu-array can be placed as document JavaScript. To place one or more menu-arrays in the document JavaScript section of the PDF, insert the `\puUseMenus` command following the last `popupmenu` environment in the preamble.

```
\puUseMenus{<menu-array-names>} (2)
```

where `<menu-array-names>` is a comma-delimited list of menu-array names; for example, `\puUseMenus{myMenu,yourMenu}`.

Tracing can only occur when (1) the `popupmenu` environment appears in the preamble; (2) and the `<name>` of the pop-up menu appears as one of the arguments of `\puUseMenus`; and (3) the command `\puProcessMenu` is used at the field level to open the pop-up, more on this in [Section 4.2](#).

- **In the body of the document.** For `popupmenus` environments declared in the body of the document, the corresponding menu-arrays can still be used anywhere after the declaration. Refer to [Section 4.1](#) for details.

4. Executing a pop-up menu

Two commands used to open a pop-up menu, `\popUpMenu` and `\puProcessMenu`:

```
\popUpMenu(<name>) (for no-tracking menus)
\puProcessMenu(<name>) (for tracking menus) (3)
```

where $\langle name \rangle$ is the name given to some `popupmenu` environment, refer to [display \(1\)](#), defined earlier in the document. The first one is designed for no-tracking menus, the second is for tracking menus.

The Various menu. In subsequent sections, we'll use the following pop-up menu, which is defined in the preamble of this document.

```

1 \urlPath{\homeAtUA}{http://www.math.uakron.edu/~dpstory}
2 \urlPath{\homeAeB}{http://www.acrotex.net}
3 \urlPath{\blogAeB}{http://blog.acrotex.net}
4 \urlPath{\urlCTAN}{https://www.ctan.org}
5 \urlPath{\embedYT}{http://www.youtube.com/embed}
6 \urlPath{\watchYT}{http://www.youtube.com/watch?v}
7 \begin{popupmenu}{Various}
8   \item{title=Various AcroTeX Links,return=none} % return value of 'none'
9   \item{title=-}
10  \begin{submenu}{title=AeB at U of Akron}
11    \item{title=Home page,return=\homeAtUA/acrotex.html}
12    \item{title=Tutorials,return=\homeAtUA/acrotex.html\#educational}
13  \end{submenu}
14  \begin{submenu}{title=Commercial AcroTeX}
15    \item{title=AcroTeX main page,return=\homeAeB}
16    \item{title=AcroTeX blog,return=\blogAeB}
17  \end{submenu}
18  \begin{submenu}{title=AcroTeX on CTAN}
19    \item{title=Contributions: AcroTeX,
20      return=\urlCTAN/author/story}
21    \item{title=The popupmenu Package,
22      return=\urlCTAN/pkg/popupmenu}
23  \end{submenu}
24  \begin{submenu}{title=YouTube Videos}
25    \begin{submenu}{title=Action Videos}
26      \item{title=Kung-Fu fighting (Bruce Lee version),
27        return=\embedYT/GZ9e3Dy7obA}
28      \item{title=Rocket Jump,return=\embedYT/7XzdZ4KcI8Y}
29    \end{submenu}
30    \begin{submenu}{title=Miscellaneous}
31      \item{title=J\"{u}rgen's favorite song,
32        return={\watchYT=mLDF5MBMWHE}}
33      \item{title=\Esc"Sea Hunt\Esc" US TV series (1958-61) lead-in,
34        return=\embedYT/Lz0aMoWh8Q4}
35      \item{title=Learn \cs{LaTeX} in one video,
36        return=\embedYT/VhmkLrOjLsw}
37    \end{submenu}
38  \end{submenu}
39 \end{popupmenu}

```

Notes:

- `\urlPath` • In lines (1)-(6), several URLs are declared using `\urlPath`, which is defined in `popupmenu` package.

- Line (8) The 'none; return value is used.
- Line (12) The fragment (#) is escaped (\#).
- Line (31) The value of the `title` key is passed through the `hyperref` command `\pdfstringdef`, consequently, you can use standard \LaTeX markup for Latin-1 characters.
- Line (32) The return value has an equal sign (=), the return value is enclosed in braces to avoid a `xkeyval` parsing error.
- `\Esc` • Line (33) The double quote needs to be escaped (because ultimately, the value will appear within double quotes. We use a special `\Esc` command of `popupmenu`.
- `\cs` • Line (35) To place a backslash('\'), use the `\cs` command.

The `popupmenu` can be placed in the preamble or in the body of the document. Let's begin with the one declared in the body.

4.1. Declaring `popupmenu` in the body

Here, in the body, we declare a (simple) menu:

```

❶ \begin{popupmenu}{LocalMenu}
   \item {title=First Item}
   \item {title=Second Item}
 \end{popupmenu}
 \pushButton[\CA{My Menu}\AAmouseenter{%
❷ \LocalMenu\r // Expand the command version of the menu-array
❸ var cChoice = \popUpMenu(LocalMenu);\r // use \popUpMenu
   if ( cChoice != null )
     app.alert("You chose the \""+cChoice+"\"");
   }]{LocalMenuBtn}{11bp}

```

In line ❶ we declare our simple menu. In line ❷ we expand the command version of the `menu-array`. (Refer to comment ❶ of [display \(1\)](#) on page 4.) Finally, in line ❸, we execute `\popUpMenu(LocalMenu)` (See [display \(3\)](#) on page 5, and the comments that follow). Using this technique, there is *no tracking*; that is, the menu item chosen is not checked.

The `popupmenu` can be declared in the preamble to obtain the same results, but still no tracking. To obtain tracking of the menu items, you must (1) declare `popupmenu` in the preamble; (2) include its name (`LocalMenu`) in the list of `<menu-array-names>` of the `\puUseMenus` command of [display \(2\)](#); (3) delete line ❷; (4) replace `\popUpMenu` in line ❸ with `\puProcessMenu`; and (5) the tracking option must be specified. Details of setting up tracking are found in the [Section 4.2](#).

4.2. Declaring `popupmenu` in the preamble

A `popupmenu` environment can be declared anywhere *before* its first use in field JavaScript to actually display the menu to the user; however, to obtain tracking of the menu items chosen you must (1) specify `\usepackage[tracking]{popupmenu}`, the

tracking option; (2) declare the menu (that is, the `popupmenu` environment) in the preamble; (3) list the menu name amongst the arguments of the `\puUseMenus` command; (4) use `\puProcessMenu` in lieu of `\popUpMenu` in the field JavaScript.

A bare-bones push button is as follows:

```
\pushButton[⟨KV-pairs⟩\AAmouseenter{%
  var cChoice = \puProcessMenu(⟨name⟩);\r
  if ( cChoice != null ) ⟨some-action⟩
}]⟨btnName⟩{⟨wd⟩}{⟨ht⟩}
```

For example, make a selection:

The verbatim listing of the first push button follows:

```
\pushButton[⟨CA{My Menu}BC⟩\WO\S{S}\H{N}\AAmouseenter{%
  var cChoice = \puProcessMenu(Various);\r
  if ( cChoice != null ) {\r\t
    if (PUdebug)\r\t\t
      app.alert("URL: \\""+cChoice+"\\"");\r\t
    else app.launchURL(cChoice);\r
  }]{mymenu}{11bp}
```

This push button references the `Various` menu, tediously listed on page 6. The second button, labeled `Action`, allows you to play around with the pop-up menu without going to the web sites. Click on it, and the caption now says `Debug`. Now, instead of going to the web site, an alert box appears announcing your choice. Good for testing things.

Multiple action types. In all the examples of this document, as well as the demo files, all actions are the same, either the return is a URL and the action is `app.launchURL(URL)` or the return is text and results are reported in an alert box. You can have multiple action types, as is illustrated in the following local declaration.

Pick your choice: . The verbatim list is,

```
% Use the defineJS environment to define the action
\begin{defineJS}[⟨makeesc⟩]{\puMultiActn}
|puMulti
var cChoice = |popUpMenu(puMulti);
if ( cChoice != null ) {
  switch (cChoice) {
    case "0":
      app.alert("You chose Item 1 from the menu");break;
    case "1":
      app.launchURL("|homeAeB");break;
    default: console.show();
      console.println("Menu returned a value of \\""+cChoice+"\\"");
      break;
  }
}
\end{defineJS}
```



```

% Declare popupmenu environment, return values are integers
\begin{popupmenu}{puMulti}
  \item {title=Item 1,return=0}
  \item {title=Item 2,return=1}
  \begin{submenu}{title=Other items}
    \item{title=Item 3,return=2}
  \end{submenu}
\end{popupmenu}
% Now execute the pop-up menu as a mouse enter event
\pushButton[\CA{Multi}\Amouseenter{\puMultiActn}]{MultiBtn}{{}{11bp}

```

5. Remarks on ps2pdf

To use the dvips->ps2pdf workflow, document JavaScript needs to be avoided. Do not use the tracking option, do not use `\puUseMenus`, and use only the `\popUpMenu` command in field JavaScript. A simple outline of a document is found below, it is a working example.

```

\documentclass{article}
\usepackage{popupmenu}
\parindent0pt\parskip6pt
\begin{document}
% Declare in the body of the text, can use anywhere after this declaration
\begin{popupmenu}{LocalMenu}
  \item {title=First Item}
  \item {title=Second Item}
\end{popupmenu}

```

```

Pop-up menu using \verb|\pushButton| of \textsf{eforms}:
\pushButton[\CA{My Menu}\Amouseenter{\LocalMenu\r
var cChoice = \popUpMenu(LocalMenu);\r
if ( cChoice != null )\r\t
  app.alert("You chose the \""+cChoice+"\" menu item");
}]{LocalMenuBtn}{{}{11bp}.

```

```

Pop-up menu using \verb|\PushButton| of \textsf{hyperref}:
\PushButton[name=hyperbutton1,onenter={\LocalMenu
var cChoice = \popUpMenu(LocalMenu);
if ( cChoice != null )
  app.alert("You chose the \""+cChoice+"\" menu item");}
]{My Menu}
\end{document}

```

Back to my retirement. 