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 popupmenu environment 3  

3 The popupmenu environment 4  

4 Executing a pop-up menu 5  
4.1 Declaring popupmenu in the body  .................................. 7  
4.2 Declaring popupmenu in the preamble 7  

5 Remarks on ps2pdf 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: Info. 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 `Info` 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, `Info`, 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.\footnote{http://www.ctan/pkg/acrotex}

Workflows. This is a general \LaTeX{} package, any workflows can be used to build a popupmenu document: pdflatex, lualatex, xelatex, or dvips -> ⟨distiller | ps2pdf⟩.\footnote{popupmenu uses document JavaScript when \texttt{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.}

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 \texttt{\ Intro} that expands to what you see below.
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. Finally, if nonempty, an alert dialog box is emitted with the return value displayed in it. BTY, \popUpMenu expands to the JavaScript method app.popUpMenuEx().

3. The popupmenu environment

This general syntax for the popupmenu environment is,

\begin{popupmenu}{⟨name⟩}
\item{title=⟨str⟩,marked=⟨true|false⟩,enabled=⟨true|false⟩,return=⟨str⟩}
\item{title=-}
...
\begin{submenu}{title=⟨str⟩,marked=⟨true|false⟩}
\item{title=⟨str⟩,marked=⟨true|false⟩,enabled=⟨true|false⟩,return=⟨str⟩}
\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 \langle 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 {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⟩} \hfill (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)} \hfill (for no-tracking menus)
\puProcessMenu{(name)} \hfill (for tracking menus) \hfill (3)
where \textit{name} is the name given to some \texttt{popupmenu} environment, refer to \texttt{display} (1), defined earlier in the document. The first one is designed for no-tracking menus, the second is for tracking menus.

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

\begin{verbatim}
\item{title=Various AcroTeX Links,return=none} % return value of 'none'
\item{title=-}
\begin{submenu}{title=AeB at U of Akron}
\item{title=Home page,return=\url{http://www.math.uakron.edu/~dpstory}}
\item{title=Tutorials,return=\url{http://www.math.uakron.edu/~dpstory/educational}}
\end{submenu}
\begin{submenu}{title=Commercial AcroTeX}
\item{title=AcroTeX main page,return=\url{http://www.acrotex.net}}
\item{title=AcroTeX blog,return=\url{http://www.acrotex.net}}
\end{submenu}
\begin{submenu}{title=AcroTeX on CTAN}
\item{title=Contributions: AcroTeX, return=\url{https://www.ctan.org/author/story}}
\item{title=The \texttt{popupmenu} Package, return=\url{https://www.ctan.org/pkg/popupmenu}}
\end{submenu}
\begin{submenu}{title=YouTube Videos}
\item{title=Action Videos}
\begin{submenu}{title=Kung-Fu fighting (Bruce Lee version), return=\url{http://www.youtube.com/embed/GZ9e3Dy7obA}}
\item{title=Rocket Jump,return=\url{http://www.youtube.com/watch?v=7XzdZ4KcI8Y}}
\end{submenu}
\item{title=J"{u}rgen's favorite song, return={\url{http://www.youtube.com/watch?v=mLDF5MBMWHE}}}  \\
\item{title="Sea Hunt" US TV series (1958-61) lead-in, return=\url{http://www.youtube.com/watch?v=Lz0aMoWh8Q4}}
\item{title=Learn \texttt{\LaTeX} in one video, return=\url{http://www.youtube.com/watch?v=VhmkLrOjLsw}}
\end{submenu}
\end{submenu}
\end{popupmenu}
\end{verbatim}

Notes:

- In lines (1)–(6), several URLs are declared using \texttt{\urlPath}, which is defined in \texttt{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.

• 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.

• 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: My Menu

1 \begin{popupmenu}{LocalMenu}
   \item {title=First Item}
   \item {title=Second Item}
\end{popupmenu}

\pushButton[CA{My Menu}AAmouseenter{%
2 \LocalMenu\r // Expand the command version of the menu-array
\begin{Verbatim}
\end{Verbatim}
3 var cChoice = \popUpMenu(LocalMenu);\r // use \popUpMenu
   if ( cChoice != null )
      app.alert("You chose the \""+cChoice+\"\"");
\end{Verbatim}
\end{Verbatim}

In line 1 we declare our simple menu. In line 2 we expand the command version of the menu-array. (Refer to comment 1 of display (1) on page 4.) Finally, in line 3, 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 \langle menu-array-names \rangle of the \puUseMenus command of display (2); (3) delete line 2; (4) replace \popUpMenu in line 3 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
Executing a pop-up menu

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

A bare-bones push button is as follows:
\pushButton[(KV-pairs)\Amouseenter{%
    var cChoice = \puProcessMenu(name);\r
    if ( cChoice != null ) (some-action)
}]{btnName}{(wd)}{(ht)}

For example, make a selection: My Menu Action

The verbatim listing of the first push button follows:
\pushButton[CA(My Menu)BC{}\W0\S(S)H(N)\Amouseenter{%
    var cChoice = \puProcessMenu(Various);\r
    if ( cChoice != null ) {\r\t
        if (Pdebug)\r\t\t
            app.alert("URL: "+cChoice+"\r;\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: Multi. 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+"\r");
            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}\AAmouseenter{puMultiActn}]{MultiBtn}{}

5. Remarks on \texttt{ps2pdf}

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

\documentclass{article}
\usepackage{popupmenu}
\parindent0pt\parskip6pt
\begin{document}
  \begin{popupmenu}{LocalMenu}
    \item {title=First Item}
    \item {title=Second Item}
  \end{popupmenu}

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

  Pop-up menu using \texttt{\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. ☻