The popupmenu Package

D. P. Story
Email: dpstory@acrotex.net

processed July 29, 2020

Contents


```
1 ⟨*package⟩
```

1 Instroduction

This is a short package that provides environments and commands for building a popup menu using JavaScript. The command `\popUpMenu` uses the Acrobat JavaScript method `app.popUpMenuEx`. This latter method requires you to pass to it a structured menu listing of the menu items to be displayed in the popup menu, and the actions to be taken when a menu item is selected. The environments `popupmenu` and `submenu` are defined for the purpose of creating this hierarchical structure.

2 Package Options

```
2 \RequirePackage{xkeyval}
3 \newif\iftrackingPU \trackingPUfalse
4 \DeclareOptionX{tracking}{\trackingPUtrue\def\puTracking{true}}
5 \DeclareOptionX{!tracking}{\trackingPUfalse\def\puTracking{false}}
6 \def\puTracking{false}
7 \ProcessOptionsX\relax
8 \edef\pu@restoreCats{%
9   \catcode`\noexpand\"=\the\catcode`\"\relax
```

```
  \catcode`\noexpand\'=\the\catcode`\'\relax
  \catcode`\noexpand\,=\the\catcode`\,\relax
  \catcode`\noexpand\!=\the\catcode`\!\relax
}
\@makeother\"\@makeother\'\@makeother\,\@makeother\!
```

3 Required packages

```
\RequirePackage{eforms}
```

4 The popupmenu environment

According to the JavaScript manual, the `app.popUpMenuEx` method takes one or more `MenuItem` objects. The LaTeX access to the properties of this object are documented as follows. This set of keys becomes the xkeyval family `menustruct` of keys for this package:

`title=`⟨*string*`|-`⟩ 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.

`marked=`⟨`true|false`⟩ (optional) A Boolean value specifying whether the item is to be marked with a check. The default is `false` (not marked).

`enabled=`⟨`true|false`⟩ (optional) A Boolean value specifying whether the item is to appear enabled or grayed out. The default is `true` (enabled).

`return=`⟨*string*⟩ (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.

```
\def\title@dash{-}\def\puNone{none}
\define@key{menustruct}{title}[]{\Hy@unicodefalse
  \let\btitle@dash\ef@NO
  \def\@rgi{#1}\ifx\@rgi\title@dash\let\btitle@dash\ef@YES\fi
  \pdfstringdef\menustruct@title{#1}}
\define@boolkey{menustruct}{marked}[true]{}
\define@boolkey{menustruct}{enabled}[true]{}
\define@key{menustruct}{return}[]{\def\menustruct@return{#1}\relax
  \ifx\menustruct@return\puNone\def\menustruct@return{null}\fi}
```

We use the command `\pum@holdtoks` to hold the menu items as they are processed in the environment, and use `\@AddToMenuToks` to add to the items.

```
\let\pum@holdtoks\@empty
\let\pum@holdtoksEx\@empty
\def\@AddToMenuToks{\g@addto@macro\pum@holdtoks}
\def\@AddToMenuToksEx{\g@addto@macro\pum@holdtoksEx}
```

`popupmenu{`⟨*name*⟩`}` The ⟨*name*⟩ argument should be consist of letters only, for ⟨*name*⟩ will be made into the command \⟨*name*⟩. The ⟨*name*⟩ has a duel rule, \⟨*name*⟩ is a macro that expands to a JavaScript array of menu items; and the name itself ⟨*name*⟩ is the name of a JavaScript variable. We begin by defining our menu structure using the `popupmenu` environment. Within this environment, we list the items in the menu using `\item` and the `submenu` menu if there are sub-menus.

The popupmenu command requires one parameter, this command is used to create both a command and a JavaScript variable. The name is passed to the \popUpMenu command, while the command version of the name expands to the menu structure (an array).

There are two ways of passing the array that is the menu structure to \popUpMenu:

1. From the document level: The arrays are declared at the document level, the name of the array is passed as the argument of \popUpMenu(⟨*menu-array*⟩).

2. From the field level: Within the script for a push button, for example, use the command version of the array name to expand first, then it can be referenced.

```
\urlPath{\aebhome}{http://www.math.uakron.edu/~dpstory}
\begin{popupmenu}{myMenu}
   \item{title=AeB,return=\aebhome/webeq.html}
   \item{title=-}
   \begin{submenu}{title=AeB Pro Family}
       \item{title=Home page, return=\aebhome/aeb_pro.html}
       \item{title=Graphicxsp, return=\aebhome/graphicxsp.html}
   \end{submenu}
   \item{title=eqExam, return=\aebhome/eqexam.html}
\end{popupmenu}
\puUseMenus{myMenu} % preamble
```

The \puUseMenus declares that myMenu is to be embedded in the PDF as document JavaScript. If \puUseMenus is not expanded in the preamble The above definition can be conveniently placed in the preamble, though it can appear anywhere before it is used, obviously. Now to use the menu structure, all we need is a push button or link to create a JavaScript action:

```
\pushButton[\CA{Packages}\AAmouseenter{%
   var cChoice = \popUpMenu(myMenu);\r
   if ( cChoice != null ) app.launchURL(cChoice);
}]{menu}{}{11bp}
```

The above example uses the eforms package, but a push button from hyperref will do too. The app.popUpMenuEx method returns the return value, which we, in turn, process. In this case, the return is a URL, which we launch.

Now, if we did not place \puUseMenus{myMenu} in the preamble, it can be used at the field level. The push button above would then need to be,

```
\pushButton[\CA{Packages}\AAmouseenter{%
   \myMenu\r
   var cChoice = \popUpMenu(myMenu);\r
   if ( cChoice != null ) app.launchURL(cChoice);
}]{menu}{}{11bp}
```

Also, in the above example, you see how the name, `myMenu`, passed as an argument of the popupmenu environment is used as a name and as a command: The name is passed to `\popUpMenu`, while the command expands to the menu structure that is referenced by the name.

`\itemindex` We generate the index of each menu item. `\itemindex` is the index of the menu structure array; for example, `\itemindex` might expand to `[0]`, `[1].oSubMenu[3]`, or `[2].oSubMenu[3].oSubMenu[0]`. If `\itemindex` is included in the return value (possibly as an array entry), we can know the item the user selected.

```
var aChoice=processMenu(AeBMenu);
if (aChoice!=null) {
    var thisChoice=aChoice[0]; // this is a string
    var thistitle=eval("AeBMenu"+thisChoice+".cName");
    app.alert(thistitle);
}
```

The above code gets the return array, then uses it to get the title of the item selected.

```
\newcount\pum@cnt
\def\pum@updateindex{\global\advance\pum@cnt\@ne
   \edef\pum@rc{\pum@topindex[\the\pum@cnt]}\edef\itemindex{'\pum@rc'}}
\def\pum@initIndexMenu#1{\global\pum@cnt=\m@ne\edef\pum@rc{#1}%
   \edef\pum@topindex{\pum@rc}}
```

We are now ready to define the `popupmenu` environment. The environment takes one required parameter, a name that is used as a JavaScript variable. This name is also used to create a command.

```
\newcount\submenuLevel \submenuLevel\z@
\newenvironment{popupmenu}[1]{\pum@initIndexMenu{}\submenuLevel\z@
  \ifpdfmarkup
    \def\textbackslash{\eqbs}\relax
    \def\Esc{\textbackslash}\relax
    \def\csiv{\eqbs\eqbs}\relax
    \def\cs##1{\csiv\csiv##1}\else
    \def\Esc{\eqbs\eqbs}\def\cs{\Esc\Esc}\fi
  \let\pum@holdtoks\@empty\let\pum@holdtoksEx\@empty
  \toks@={\pum@mytab}\@temptokena={\pum@mytab}\@makeother\~%
```

We initialize with a `\@gobble`, which eats up the leading comma (,) that is placed there by the code below.

```
  \gdef\msarg{#1}\gdef\msargEx{#1Ex}\@AddToMenuToks{\@gobble}%
  \@AddToMenuToksEx{\@gobble}\let\item\pum@item
\ignorespaces}{%
  \csarg\xdef{\msarg}{var \msarg\space = [ \pum@holdtoks^^J];}%
  \iftrackingPU
  \csarg\xdef{\msargEx}{var \msargEx\space = [ \pum@holdtoksEx^^J];}\fi
\aftergroup\ignorespaces}
```

\item{title=⟨*string*⟩,marked=⟨true|false⟩,enabled=⟨true|false⟩,return=⟨*string*⟩}

Below is the definition of \pum@item, at the startup of the popupmenu environment, we \let\item\pum@item. The definition of \pum@item takes one argument, the properties described above.

\pum@item

```
\newcommand{\pum@item}[1]{\pum@updateindex
%  \edef\tmp@exp{\noexpand
%  \setkeys{menustruct}{title,marked=false,enabled,return,#1}}\tmp@exp
  \setkeys{menustruct}{title,marked=false,enabled,return,#1}\relax
  \ifx\menustruct@title\@empty
    \PackageWarning{popupmenu}
    {A value of the key 'title' is required,\MessageBreak
      putting in a place keeper title}%
    \def\menustruct@title{This title is undefined}\fi
  \edef\tmp@exp{,^^J\the\toks@
    {cName: "\menustruct@title"%
    \ifKV@menustruct@marked, bMarked: true\fi%
    \ifKV@menustruct@enabled\else, bEnabled: false\fi%
%    \ifx\btitle@dash\ef@NO, cItem: \itemindex\fi%
    \ifx\menustruct@return\@empty\else,%
      cReturn: "\menustruct@return"\fi}}\expandafter
  \@AddToMenuToks\expandafter{\tmp@exp}%

  \edef\tmp@expEx{,^^J\the\@temptokena
    {cName: "\menustruct@title"%
    \ifKV@menustruct@marked, bMarked: true\fi%
    \ifKV@menustruct@enabled\else, bEnabled: false\fi%
%    \ifx\btitle@dash\ef@NO, cItem: \itemindex\fi%
    \ifx\btitle@dash\ef@NO
    \ifx\menustruct@return\@empty,%
      cReturn:"[\itemindex,'\menustruct@title']"%
    \else,cReturn:"[\itemindex,'\menustruct@return']"\fi\fi}}%
  \expandafter\@AddToMenuToksEx\expandafter{\tmp@expEx}%
\ignorespaces}
```

Some technical matters, we need unmatched braces, so we define \pum@lbrace and \pum@rbrace.

```
\begingroup
\catcode`<=1 \catcode`\>=2 \@makeother\{ \@makeother\}
\gdef\pum@lbrace<{>\gdef\pum@rbrace<}>
\endgroup
\def\pum@mytab{\space\space\space\space}
```

submenu{title=⟨*title*⟩,marked=⟨true|false⟩}

Used to create a submenu of a menu item. The top level menu item has no return value, it can be marked but cannot be disabled (enabled=false).

The argument of submenu are any of the menu item properties, however, only title and marked will be recognized.

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.

```
\newenvironment{submenu}[1]{\pum@updateindex\advance\submenuLevel\@ne
  \csarg\xdef{pum@cntLevel\the\submenuLevel}{\the\pum@cnt}%
%  \xdef\saved@pum@cnt{\the\pum@cnt}\relax
  \pum@initIndexMenu{\pum@rc.oSubMenu}\edef\temp@toks{\the\toks@}%
  \def\temp@toksEx{\the\@temptokena}%
  \toks@=\expandafter{\temp@toks\pum@mytab}%
  \@temptokena=\expandafter{\temp@toksEx\pum@mytab}%
  \setkeys{menustruct}{title,marked=false,enabled,return,#1}%
  \edef\tmp@exp{,^^J\the\toks@
    \noexpand\pum@lbrace cName: "\menustruct@title"%
    \ifKV@menustruct@marked, bMarked: true\fi%
    \ifKV@menustruct@enabled\else, bEnabled: false\fi,
  oSubMenu:^^J\the\toks@[]%
```

Again, we \@gobble up the leading comma (,).

```
  \expandafter\@AddToMenuToks\expandafter{\tmp@exp\@gobble}%
  \edef\tmp@expEx{,^^J\the\@temptokena
    \noexpand\pum@lbrace cName: "\menustruct@title"%
    \ifKV@menustruct@marked, bMarked: true\fi%
    \ifKV@menustruct@enabled\else, bEnabled: false\fi,
  oSubMenu:^^J\the\@temptokena[]%
  \expandafter\@AddToMenuToksEx\expandafter{\tmp@expEx\@gobble}%
\ignorespaces}{%
  \edef\tmp@exp{^^J\the\toks@ ]\pum@rbrace}%
  \edef\tmp@expEx{^^J\the\@temptokena ]\pum@rbrace}%
  \expandafter\@AddToMenuToks\expandafter{\tmp@exp}%
  \expandafter\@AddToMenuToksEx\expandafter{\tmp@expEx}%
  \global\pum@cnt\@nameuse{pum@cntLevel\the\submenuLevel}%
\aftergroup\ignorespaces}
```

\popUpMenu(⟨*name*⟩) The \popUpMenu command takes one argument, the ⟨*name*⟩ that had earlier been passed to a popupmenu environment. The command expands to the app.popUpMenuEx method. The document author must then process the return value in some way. The argument is enclosed in parentheses, this is so we can use \popUpMenu at the document level, we can pass it an argument there.

```
\def\popUpMenu(#1){app.popUpMenuEx.apply( app, #1 )}
```

\puProcessMenu(⟨*name*⟩) When the tracking option is taken, use the \puProcessMenu command to execute a menu item with tracking. If tracking is not in effect, \puProcessMenu is the same as \popUpMenu.

```
\def\puProcessMenu(#1){\iftrackingPU
  puProcessMenu("#1")\else\popUpMenu(#1)\fi}
```

\urlPath{⟨*path*⟩} A convenience command to save a url path. The string is normalized using the hyperref command \hyper@normalise. Though we don't require any other packages, you can't do much unless you use hyperref.

```
\providecommand{\urlPath}[1]{\def\pum@urlName{#1}%
```

```
  \hyper@normalise\pum@urlPath}
%\def\pum@urlPath#1{\csarg\xdef\pum@urlName{#1}}
\def\pum@urlPath#1{\expandafter\xdef\pum@urlName{#1}}
```

\puUseMenus⟨*list-arrays*⟩, where ⟨*list-arrays*⟩ is a comma-delimited list of ⟨*name*⟩s that have been declared earlier as an argument of a popupmenu environment. The arrays listed in ⟨*list-arrays*⟩ will be defined at the document level.

```
\def\puUseTheseMenus{// No pop-up data defined^^J}
\let\puMenuCmds\@empty
\newcommand{\puUseMenus}[1]{\bgroup
  \@for\pu@menu:=#1\do{\ifx\puMenuCmds\@empty
    \def\puUseTheseMenus{// popupmenu: Begin popup menu data^^J}\fi
    \expandafter\g@addto@macro\expandafter
      \puMenuCmds\expandafter{%
      \csname\pu@menu\endcsname^^J}\relax
    \iftrackingPU
      \expandafter\g@addto@macro\expandafter
      \puMenuCmds\expandafter{%
      \csname\pu@menu Ex\endcsname^^J}\relax
    \fi
    \edef\x{\expandafter\noexpand\@nameuse{\pu@menu}}%
    \toks@=\expandafter{\x^^J}%
    \expandafter\g@addto@macro\expandafter
      \puUseTheseMenus\expandafter{\the\toks@}%
    \iftrackingPU
      \edef\x{\expandafter\noexpand\@nameuse{\pu@menu Ex}}%
      \toks@=\expandafter{\x^^J}%
      \expandafter\g@addto@macro\expandafter
        \puUseTheseMenus\expandafter{\the\toks@}%
    \fi
  }\g@addto@macro\puUseTheseMenus
    {// popupmenu: End of popup menu data}\egroup
  \ifx\puUseTheseMenus\@empty
  \def\puUseTheseMenus{// No pop-up data defined}\fi
}
```

A small insDLJS environment to create the menu arrays at the document level. The command \puUseTheseMenus will expand to the array declarations.

```
\iftrackingPU
\begin{insDLJS}{pujs}{Pop-up Menu Data}
\puUseTheseMenus
\end{insDLJS}
\@onlypreamble\puUseMenus
\begin{insDLJS*}{pumenu}
\begin{newsegment}{popupmenu: Menu tracking support}
var trackingPU=\puTracking;
var PUdebug=false;
var aPULastChoice=new Array;
```

```
var bPULastChoice=false;
var bIsMarked=false;
%var aChoice;  // make local
function puProcessMenu(cMenu) { // aMenu->cMenu now a string
  var cMenuEx=(trackingPU)?cMenu+"Ex":cMenu;
  var aMenuEx=eval(cMenuEx);
  var cChoice = app.popUpMenuEx.apply( app, aMenuEx );
  if (trackingPU) {
    if ( cChoice != null ) {
      aChoice=eval(cChoice);
      if (aChoice[1]==""||aChoice[1]=="null") return null;
      var puRtn=aChoice[1];
      var thisChoice=aChoice[0];
//      eval(cMenuEx+thisChoice).bMarked=true;
      if (!bPULastChoice) {
        eval(cMenuEx+aChoice[0]).bMarked=true;
      } else {
        var structLoc=aPULastChoice[1];
        if(aPULastChoice[0]+structLoc==cMenuEx+aChoice[0]) {
          bIsMarked = eval(cMenuEx+aChoice[0]).bMarked;
          eval(cMenuEx+aChoice[0]).bMarked=!(bIsMarked);
          bPULastChoice=false;
          if (bIsMarked) var puRtn=null
  } else {
        eval(aPULastChoice[0]+structLoc).bMarked=false;
        eval(cMenuEx+aChoice[0]).bMarked=true;
  }
    }
      aPULastChoice=[cMenuEx,aChoice[0]];
      bPULastChoice=true;
      return puRtn;
    } else return null;
  } else return cChoice;
}
\end{newsegment}
\end{insDLJS*}
\fi
\pu@restoreCats
⟨/package⟩
```

5 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

6 Change History