The popupmenu Package

D. P. Story
Email: dpstory@acrotex.net
processed September 27, 2010

Contents

1 ⟨∗package⟩
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 \RequirePackage{xkeyval}
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 (taken verbatim from the JavaScript reference):

| 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. |
| marked | (optional) A Boolean value specifying whether the item is to be marked with a check. The default is false (not marked). |
| enabled | (optional) A Boolean value specifying whether the item is to appear enabled or grayed out. The default is true (enabled). |
| return | (optional) A string to be returned when the menu item is selected. The default is the value of cName. |

3 \define@key{menustruct}{title}\[\{\def\menustruct\@title{#1}\}\]
4 \define@boolkey{menustruct}{marked}\[true\]\[\{}\]
5 \define@boolkey{menustruct}{enabled}\[true\]\[\{}\]
6 \define@key{menustruct}{return}\[\{}\]
7 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.
8 \let\pum@holdtoks@empty
9 newcommand{\@AddToMenuToks}{\g@addto@macro\pum@holdtoks}

1
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. The menu structure can be placed at the document level, or as part of a push button action. Here is an example of usage:

```latex
\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}
```

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:

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

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.

If we have placed `\myMenu` at the document level, the line `\myMenu\r` would not be needed. If you are using the same menu several times in the document, put it at the document level to reduce file size.

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.

```latex
\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
```

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

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

```
9 \newcount\pum@cnt
10 \edef\pum@updateindex{\global\advance\pum@cnt\@one
11 \edef\pum@rc{\pum@topindex[\the\pum@cnt]}\edef\itemindex{\'\pum@rc'}
12 \def\pum@initIndexMenu#1{\global\pum@cnt=-1\relax\def\pum@rc{#1}%
13 \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.

```
14 \newenvironment{popupmenu}[1]{\pum@initIndexMenu{}%
15 \let\pum@holdtoks@empty
16 \toks@={\pum@mytab}\@makeother\˜
We initialize with a \@gobble, which eats up the leading comma (,) that is placed there by the code below.
17 \gdef\msarg{#1}\@AddToMenuToks{\@gobble}%
18 \let\item\pum@item
19 }%
20 \expandafter\xdef\csname\msarg\endcsname{[
21 \var\msarg\space=\[\pum@holdtoksˆˆJ\];}%
22 }
```

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

```
23 \newcommand{\pum@item}[1]{\pum@updateindex
24 \edef\tmp@exp{\noexpand
25 \setkeys{menustruct}{title,marked=false,enabled,return,#1}}\tmp@exp
26 \edef\tmp@exp{,ˆˆJ\the\toks@}
27 \cName: "\menustruct@title"
28 \text{ifKV\menustruct@marked, bMarked: true\fi%
29 \text{ifKV\menustruct@enabled\else, bEnabled: false\fi%
30 \text{ifKV\menustruct@return\empty\else,
31 cReturn: "\menustruct@return"\fi}}%
32 \expandafter\@AddToMenuToks\expandafter{\tmp@exp}%
33 }
```

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

```
34 \begingroup
35 \catcode`<=1 \catcode`>=2 \@makeother{\@makeother}
36 \edef\pum@lbrace<\gdef\pum@rbrace>
37 \endgroup
38 \def\pum@mytab{\space\space\space\space}
39 ```
submenu 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.

\def\submenu[#1]{\pum@updateindex
  \xdef\saved@pum@cnt{\the\pum@cnt}\
  \pum@initIndexMenu{\pum@rc.oSubMenu}\edef\temp@toks{\the\toks@}\
  \toks@=\expandafter{\temp@toks\pum@mytab}%
  \setkeys{menustruct}{title,marked=false,enabled,return,#1}%
  \edef\tmp@exp{,\the\toks@}%
  \noexpand\pum@lbrace cName: "\menustruct@title",%
  \ifKV@menustruct@marked, bMarked: true\fi,%
  \ifKV@menustruct@enabled\else, bEnabled: false\fi,%
  oSubMenu: \the\toks@}%
Again, we \@gobble up the leading comma (,).
\expandafter\@AddToMenuToks\expandafter{\tmp@exp\@gobble}%
}\%
\expandafter\@AddToMenuToks\expandafter{\tmp@exp}%
\global\pum@cnt=\saved@pum@cnt
}

\popUpMenu The \popUpMenu command takes one argument, the name pass 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 )}

\urlPath 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 as well.
\providecommand{\urlPath}[1]{\def\pum@urlName{#1}\
  \hyper@normalise\pum@urlPath}\
\def\pum@urlPath#1{\expandafter{\expandafter{%}
  \pum@urlName{#1}}}

\endinput