The ltx4yt Package

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

processed June 10, 2021

Contents

1 ⟨*package⟩

1 Introduction

It is July 2020 and Adobe's support for Flash will vanish in December. Google stopped supporting Flash several years ago, so the package yt4pdf is no longer functional and will be withdrawn from CTAN. If we cannot play videos within PDF anymore, the next best course is to develop some basic commands for playing YouTube videos in the default browser, preferably without any annoying advertisements. This can be done for some videos, but for others it cannot be done.

```
2 \RequirePackage{xkeyval}
```

usepopup
!usepopup

When this option is taken, additional code to support the use of popup menus is input. We also support `!popup`, which is a convenience option for not using `usepopup`.

```
3 \DeclareOption{usepopup}{\def\lo@dpu{\InputIfFileExists{ytpu.def}
4   {\PackageInfo{ltx4yt}{Loading ytpu.def}}
5   {\PackageInfo{ltx4yt}{Can't find ytpu.def}}}}
```

```
\DeclareOption{!usepopup}{}
\let\lo@dpu\relax
\AtEndOfPackage{\lo@dpu}
\ProcessOptions
\edef\yt@restoreCats{%
  \catcode`\noexpand\"=\the\catcode`\"\relax
  \catcode`\noexpand\'=\the\catcode`\'\relax
  \catcode`\noexpand\,=\the\catcode`\,\relax
  \catcode`\noexpand\!=\the\catcode`\!\relax
}
\@makeother\"\@makeother\'\@makeother\,\@makeother\!
\RequirePackage{xcolor}
\RequirePackage{eforms}
```

If the usepopup option is taken, we load the popupmenu package.

```
\ifx\lo@dpu\relax\else
\def\YT@rpPU{\RequirePackage{popupmenu}[2020/07/26]}\expandafter
\YT@rpPU\fi
```

2 Implementation

The JavaScript method `app.launchURL{`⟨*URL*⟩`}` is used to open the default browser on the page determined by ⟨*URL*⟩. The links created use `\URI` (`/S/URI/URI`), this action type allows links to be functional in all PDF viewers, even on hand held devices.

We have three implementations of this plan: (1) links (`\ytvId` and `\ytLink`) for all devices; (2) dropdown menus (combo boxes) (`\ytComboList` (uses JavaScript); and (3) pop-up menus using the `popupmenu` environment of the popupmenu package (uses JavaScript). The methods that use JavaScript will not function in the near future on hand held devices.

2.1 Playing a YouTube video from its Video ID

In this section, we create several ways of calling for a YouTube video to play in the default browser.

2.1.1 Using a Link

`\ytvIdPresets{`⟨*KV-pairs*⟩`}` The options for the `\ytvId` link. The default is given below in the definition. Initially, the preset is to produce `webbrown` colored links.

```
\newcommand{\ytvIdPresets}[1]{\def\yt@vIdPresets{#1}}
\definecolor{webbrown}{rgb}{.6,0,0} % from the web package
\ytvIdPresets{\linktxtcolor{webbrown}}
```

Here is a convenience command. This definition has since been made in insdljs.

```
\providecommand{\URI}[1]{/S/URI/URI(#1)}
```

`\ytURL` Define `\ytURL` to expand to the YouTube web site.

```
\def\ytNF{false}
\def\ytURL{https://www.youtube.com}
```

`\ytvId*[`⟨*opts*⟩`]{`⟨*ytvID*⟩`}{`⟨*text*⟩`}` The `\ytvId` is link which when pressed plays the video whose Video Id is ⟨*ytvID*⟩) in the default browser.

`*` If the `*`-option is taken, you have determined that this video cannot be embedded (which is the ideal) and must be played normally on YouTube with all advertisements and other extraneous information.

⟨*opts*⟩: link optional KV-pairs to modify the appearance of the link.

⟨*ytvID*⟩: The video Id for the YouTube video to play

⟨*text*⟩: The text that displays the link.

```
\def\ytvIdParams#1{\def\@rgi{#1}\ifx\@rgi\@empty
  \let\ytvIdP@rams\@empty\else\def\ytvIdP@rams{#1}\fi}
\let\ytvIdP@rams\@empty
\newcommand{\ytvId}{\@ifstar{\def\yt@ask{*}\yt@@vId}
  {\let\yt@ask\@empty\yt@@vId}}
\newcommand{\yt@@vId}[3][]{\begingroup
  \ifx\ytvIdP@rams\@empty\let\ques\@empty\else
    \ifx\yt@ask\@empty\def\ques{?}\else\def\ques{&}\fi
  \fi
  \ifx\yt@ask\@empty
    \def\yt@lnk@hash{embed/#2\ques\ytvIdP@rams}\else
    \def\yt@lnk@hash{watch?v=#2\ques\ytvIdP@rams}\fi
  \setLink[\presets{\yt@vIdPresets}#1
    \A{\URI{\ytURL/\yt@lnk@hash}}]{#3}\endgroup
}
```

`\ytvIdML*[`⟨*opts*⟩`]{`⟨*ytvID*⟩`}{`⟨*text*⟩`}` The `\ytvId` is link which when pressed plays the video whose Video Id is ⟨*ytvID*⟩) in the default browser. This is a multi-line link, it requires the aeb_mlink package and the dvips -> distiller workflow.

```
\newcommand{\ytvIdML}{\@ifstar{\def\yt@ask{*}\yt@@vIdML}
  {\let\yt@ask\@empty\yt@@vIdML}}
\newcommand{\yt@@vIdML}[3][]{\begingroup
  \ifx\ytvIdP@rams\@empty\let\ques\@empty\else
    \ifx\yt@ask\@empty\def\ques{?}\else\def\ques{&}\fi
  \fi
  \ifx\yt@ask\@empty
    \def\yt@lnk@hash{embed/#2\ques\ytvIdP@rams}\else
    \def\yt@lnk@hash{watch?v=#2\ques\ytvIdP@rams}\fi
  \mlsetLink[\presets{\yt@vIdPresets}#1
    \A{\URI{\ytURL/\yt@lnk@hash}}]{#3}\endgroup
}
```

`\ytLink[`⟨*opts*⟩`]{`⟨*spec*⟩`}{`⟨*text*⟩`}` We create a custom link for youtube videos. The ⟨*spec*⟩ (specification) argument has several forms for maximum convenience and flexibility.

- `\ytLink{\embedId{`⟨*ytvID*⟩`}\params{`⟨*parameters*⟩`}}{`⟨*text*⟩`}`

  This is the form for playing a video with ⟨*ytvID*⟩ that *can be embedded* (this best type of video). The `\params` argument must follow the `\embedId`, its argument are any parameters you want to append to the URL. The use of `\params` is optional; however, without the `\params` you may as well use `\ytvId{`⟨*ytvID*⟩`}{`⟨*text*⟩`}`.

- `\ytLink{\watchId{`⟨*ytvID*⟩`}\params{`⟨*parameters*⟩`}}{`⟨*text*⟩`}`

  This is the form for playing a video with ⟨*ytvID*⟩ that *cannot be embedded* (advertisements and other information appears). The `\params` argument must follow the `\watchId`, its argument are any parameters you want to append to the URL. The use of `\params` is optional; however, without the `\params` you may as well use `\ytvId*{`⟨*ytvID*⟩`}{`⟨*text*⟩`}`.

- `\ytLink{\embed{`⟨*spec*⟩`}}{`⟨*text*⟩`}`

  A form that does not specify a video ID. It is useful for more general actions, such as searches, for example,

  Note: `listType=search` is deprecated and will no longer be supported as of 15 November, 2020.

  ```
  \ytLink{\embed{listType=search&list=Adobe Acrobat DC}}
      {Search for Adobe Acrobat DC}
  ```

  This form does not have a `\params` optional argument as all the parameters are built into the argument of `\embed`.

- `\ytLink{`⟨*spec*⟩`}{`⟨*text*⟩`}`

  The most general form. The action of this link is `\URI{\ytURL/`⟨*spec*⟩`}`; for example,

  Note: `listType=search` is deprecated and will no longer be supported as of 15 November, 2020.

  ```
  \ytLink{embed?listType=search&list=Adobe Acrobat DC}
      {Search for Adobe Acrobat DC}
  ```

  Here you are free to build whatever URL you can imagine.

- `\ytLink{\channel{`⟨*name*⟩`}}{`⟨*text*⟩`}`

  Such a link displays the channel ⟨*name*⟩. For example,

  ```
  \ytLink{\channel{rocketjump}}{The RocketJump Channel}
  ```

  This link opens the YouTube channel named `rocketjump`.

Passing Player Parameters. As was illustrated above, the custom link `\ytLink` can pass various recognizable parameters to YouTube. After reviewing,

https://developers.google.com/youtube/player_parameters

the following parameters are recommended, some of them are illustrated in the sample document `ltx4yt-1.tex`:

- `autoplay=`⟨`0`|`1`⟩ (default `0`)
- `controls=`⟨`0`|`1`⟩ (default `1`)
- `fs=`⟨`0`|`1`⟩ (default `1`)
- `modestbranding=`⟨`0`|`0`⟩
- `playlist=`⟨*list*⟩
- `listType=search&list=`⟨*query*⟩
  Note: `listType=search` is deprecated and will no longer be supported as of 15 November, 2020.

The specialized needs of the document author is most easily accommodated through the use of the `\ytLink` command, for example,

```
\ytLink{\embedId{5y9-EVmreU4}\params{autoplay=1&modestbranding=1}}
    {Lori's Corner: Episode \#1}
```

```
\newif\ifytwatch \ytwatchfalse
```

The `\ytLink` command. Before getting to `\ytLink` and `\ytLinkML`, there is a long stream of commands to parse the ⟨*spec*⟩ argument of `\ytLink`. It goes through looking for `\watchId`, `\embedId`, `\embed`, and `\params`. As it progresses, it adds code to the macro `\ytspec`, which at the end of things will hold the fully formed ⟨*spec*⟩ for insertion into the URL.

```
\def\yt@@parse{\let\ytspec\@empty\yt@parse}
\def\yt@parse{\@ifnextchar\@nil{\@gobble}{\yt@parsei}}
\def\yt@parsei{\@ifnextchar\watchId{% check for \watchId
  \ytwatchtrue\yt@parse@watch}{\yt@parseii}}
\def\yt@parse@watch\watchId#1{\g@addto@macro
  \ytspec{\watchId{#1}}\yt@parse}
\def\yt@parseii{\@ifnextchar\embedId{% check for \embedId
  \yt@parse@embedId}{\yt@parseiii}}
\def\yt@parse@embedId\embedId#1{\g@addto@macro
  \ytspec{\embedId{#1}}\yt@parse}
\def\yt@parseiii{\@ifnextchar\embed{% check for \embed
  \yt@parse@embed}{\yt@parseiv}}
\def\yt@parse@embed\embed#1{\g@addto@macro
  \ytspec{\embed{#1}}\yt@parse}
\def\yt@parseiv{\@ifnextchar\params{% check for \params
  \yt@parse@params}{\yt@parsev}}
\def\yt@parse@params\params#1{\ifytwatch
```

```
  \g@addto@macro\ytspec{&#1}\else
  \g@addto@macro\ytspec{?#1}\fi
  \yt@parse}
```

If we get this far, ⟨*spec*⟩ is raw, has none of the helper commands. So we just insert the entire argument into `\ytspec`.

```
\def\yt@parsev#1\@nil{\g@addto@macro\ytspec{#1}}
```

All preliminaries done, we define `\ytLink`

```
\newcommand{\ytLink}[3][]{\begingroup
  \def\embedId##1{embed/##1}%
  \def\params##1{##1}\def\embed##1{embed?##1}%
  \def\watchId##1{watch?v=##1}\def\channel##1{c/##1}%
  \def\search##1{results?search_query=##1}%
  \def\user##1{user/##1}%
  \yt@@parse#2\@nil % returns arg in \ytspec
  \def\URLArg{\ytURL/\ytspec}%
  \setLink[\presets{\yt@vIdPresets}#1\A{\URI{\URLArg}}
  ]{#3}\endgroup
}
```

`\ytLinkML[`⟨*opts*⟩`]{`⟨*spec*⟩`}{`⟨*text*⟩`}` is the multi-line version of `\ytLink`. It requires a dvips -> distiller workflow as well as the aeb_mlink package.

```
\newcommand{\ytLinkML}[3][]{\begingroup
  \def\embedId##1{embed/##1}%
  \def\params##1{##1}\def\embed##1{embed?##1}%
  \def\watchId##1{watch?v=##1}\def\channel##1{c/##1}%
  \def\search##1{results?search_query=##1}%
  \def\user##1{user/##1}%
  \yt@@parse#2\@nil % returns arg in \ytspec
  \def\URLArg{\ytURL/\ytspec}%
  \mlsetLink[\presets{\yt@vIdPresets}#1
    \A{\URI{\URLArg}}
  ]{#3}\endgroup
}
```

2.1.2 Using a Combobox

All commands associate with a combo box play list.

`\ytComboList[`⟨*opts*⟩`]{`⟨*name*⟩`}{`⟨*wd*⟩`}{`⟨*ht*⟩`}` The `\ytComboList` command produces a combo box (dropdown menu) of video Ids and titles. The user selects a video based on its title, then presses the PLAY button. The two commands `\ytComboListPresets` and `\ytComboBtnPresets` are used to set the appearances of the combo box and the PLAY button.

⟨*opts*⟩: eforms key-value pairs

⟨*name*⟩: A unique name (ASCII letters and numbers)

⟨*wd*⟩: The width of the combo box

⟨*ht*⟩: The height of the combo box

Now we have the code for `\ytComboList`

```
\newcommand{\ytComboList}[4][]{%
  \comboBox[\Ff{\FfCommitOnSelChange}\DV{\yt@pl@def}\V{\yt@pl@def}
    \presets{\yt@ComboListPresets}#1]{ytSelect#2}
  {#3}{#4}{*\yt@pl@pl}% 2020/07/22 v0.4
}
```

`\ytComboBtn`[⟨*opts*⟩]{⟨*name*⟩}{⟨*wd*⟩}{⟨*ht*⟩} A button to play the selection made in the combo box. The caption of the push button is determined by the command `\ytStrPlay`. The parameters for `\ytComboBtn` are,

`\ytStrPLAY`

⟨*opts*⟩: The KV-pairs to pass to the underlying push button.

⟨*name*⟩: A unique name (ASCII letters and numbers)

⟨*wd*⟩: The width of the combo box

⟨*ht*⟩: The height of the combo box

```
\newcommand{\ytStrPLAY}{PLAY}
\newcommand{\ytComboBtn}[4][]{%
  \pushButton[\TU{Click to play}\CA{\ytStrPLAY}
    \presets{\yt@ComboBtnPresets}#1
    \A{\JS{var f=this.getField("ytSelect#2");\r
    var ytID=f.value;\r
    var i=f.currentValueIndices;\r
    var ytFV=f.getItemAt(i,false);\r
    var i=ytFV.indexOf("*");\r
    if ( i == -1 )\r\t
      app.launchURL("\ytURL/embed/"+ytID,\ytNF);\r
    else\r\t
      app.launchURL("\ytURL/watch?v="+ytID,\ytNF);
  }}]{ytSelectBtn#2}{#3}{#4}%
}
```

`\ytPlayList`{⟨*ytvID*⟩}{⟨*\cmd*⟩} This command is executed before `\ytComboList` to set the initial/default value (⟨*ytvID*⟩) of the combo box and the play list (⟨*\cmd*⟩). Here ⟨*\cmd*⟩ is a command defined by the `\declarePlayList` command. Use the `\ytPlayList` to pass the play list to the next combo box.

```
\newcommand{\ytPlayList}{\begingroup\@makeother\_\@makeother\'
    \ytPlayList@i}
\def\ytPlayList@i#1#2{\gdef\yt@pl@def{#1}\xdef\yt@pl@pl{#2}\endgroup}
```

`\ytComboListPresets`{⟨*opts*⟩} The KV-pairs for `\ytComboList`.

```
\newcommand{\ytComboListPresets}[1]{\def\yt@ComboListPresets{#1}}
\let\yt@ComboListPresets\@empty
```

`\ytComboBtnPresets`{⟨*opts*⟩} The KV-pairs for `\ytComboBtn`.

```
\newcommand{\ytComboBtnPresets}[1]{\def\yt@ComboBtnPresets{#1}}
\let\yt@ComboBtnPresets\@empty
```

`\ytIdTitle{`⟨*text*⟩`}{`⟨*VidID*⟩`}` A convenience command to lay out the playlist.

```
\newcommand{\ytIdTitle}[2]{[(#2)(#1)]}
```

`\declarePlayList` A video ID may contain characters LaTeX considers special, so we sanitize these special characters before reading in the video ID. Near as I can determine, a video id consists of 11 characters comprising combinations of letters (A-Z,a-z) numbers (0-9) and special characters underscore and hyphen (_ and -). We sanitize the last two.

```
\declarePlayList{⟨\cmd⟩}{
   \ytIdTitle{⟨text⟩}{⟨VidID⟩}
   ...
   \ytIdTitle{⟨text⟩}{⟨VidID⟩}
 }
```

The entries may also be in raw form '`[(\ameta{VidID})(\ameta{text})]`'. Note that the two arguments are enclosed in parentheses, there is a problem with parsing if ⟨*text*⟩ itself contains parentheses. Within ⟨*text*⟩ enclose matching parentheses in braces, for example,

```
\ytIdTitle{Kung-Fu Fighting {(Bruce Lee version)}}{GZ9e3Dy7obA}
```

The role of * in the title. If an * appears in the title, this means that the video cannot be embedded. Viewing the video will come with advertisements and other information that YouTube generates.

```
\ytIdTitle{Kung-Fu Fighting* {(Original version)}}{jhUkGIsKvn0}
```

```
\newcommand{\declarePlayList}[1]{\bgroup
  \Hy@unicodefalse
  \let\pl@yList\@empty
  \ifpdfmarkup
    \def\Esc{\eqbs}\else\def\Esc{}\fi
  \def\cs##1{\eqbs\eqbs##1}\relax
  \@makeother\_\@makeother\-
  \yt@declarePlayList{#1}%
}
\def\yt@declarePlayList#1#2{%
  \yt@declarePlayList@i#2\@nil\relax\relax
  \toks@=\expandafter{\pl@yList}\relax
  \xdef#1{\pl@yList}\egroup
}
\def\yt@declarePlayList@i{\@ifnextchar\@nil
  {\expandafter\@gobbletwo\@gobble}
  {\yt@declarePlayList@ii}%
}
\def\yt@declarePlayList@ii\ytIdTitle#1#2{%
  \pdfstringdef\yt@PLTitle{#1}%
  \edef\y{[(#2)(\yt@PLTitle)]}%
  \expandafter\g@addto@macro\expandafter
```

```
    \pl@yList\expandafter{\y}%
  \yt@declarePlayList@i
}
```

An example of use of `\declarePlayList`:

```
\declarePlayList{\playListii}{
  \ytIdTitle{Elfego Baca}{gRwa0MdeqVs}
  \ytIdTitle{Texas John Slaughter}{7yrk1BvtLE8}
  \ytIdTitle{Swamp Fox}{-SBPnw5riLM&NR}
  \ytIdTitle{Zorro Promo}{cKludhxEoJ0}
}
```

```
⟨/package⟩
⟨*pujs⟩
```

2.1.3 Using a popup menu

These code lines (including the document JavaScript) are only included when the `usepopup` option is taken.

`\ytUseMenus{`⟨*menu-names*⟩`}` A command used to list popupmenu data. It defines a command `\ytPopupData` that is used in the JS support for popup menus. The argument ⟨*menu-names*⟩ is a comma-delimited list of menu names defined by a `popupmenu` environment. The command needs to be in the preamble.

```
\def\ytPopupAllMenuData{// ltx4yt: Begin popup menu data^^J}%
\let\ytMenuNames\@gobble
\newcommand{\ytUseMenus}[1]{\bgroup
  \@for\yt@menu:=#1\do{%
    \edef\x{\noexpand\g@addto@macro\noexpand
      \ytMenuNames{,"\yt@menu"}}\x
    \edef\x{\expandafter\noexpand\@nameuse{\yt@menu}}%
    \toks@=\expandafter{\x^^J}%
    \expandafter\g@addto@macro\expandafter
      \ytPopupAllMenuData\expandafter{\the\toks@}%
  }\g@addto@macro\ytPopupAllMenuData
    {// ltx4yt: End of popup menu data}%
  \egroup
}
\@onlypreamble\ytUseMenus
```

`\puIdTitle` A convenience macro for entering popupmenu data for youtube videos.

```
\begin{popupmenu}{YTMenu}
  \puIdTitle{Select a YouTube Video}{}  A title has no yt Id
  \begin{submenu}{title=Music Videos}
     \puIdTitle{Kung-Fu Fighting Bruce Lee version}{GZ9e3Dy7obA}
     \puIdTitle{\string\"Sea Hunt\string\" TV serie}{MW-IZ67iADU}
     ...
   \end{submenu}
\end{popupmenu}
```

Note that we must protect the double quote.

```
%\newcommand{\puIdTitle}[2]{\item{title={#1},%
%  return={[\itemindex,'#2']}}}
\newcommand{\puIdTitle}[2]{\Hy@unicodefalse\pdfstringdef\x@YT{#1}%
  \edef\y@YT{\noexpand\item{title={\x@YT},%
  return={[\noexpand\itemindex,'#2']}}}\y@YT}
```

Some convenience commands for setting up a push button to display the popup-menu on rollover.

```
\def\ytpubtnCnt{0}
\newcommand{\ytPopupBtn}[4][]{\bgroup
  \@tempcnta\ytpubtnCnt\relax
  \advance\@tempcnta\@ne
  \xdef\ytpubtnCnt{\the\@tempcnta}%
  \pushButton[\cmd{\pmpvCAOff}\CA{YT Menu}
  \textColor{0 0 1}\W1\BC{}\textSize{0}
    \H{N}\S{S}\presets{\yt@PopupPresets}#1
    \AAmouseenter{ytPopupMenu("#2");} % dps
    ]{ytPopup\ytpubtnCnt}{#3}{#4}\egroup
}
\newcommand{\ytPopupPresets}[1]{\def\yt@PopupPresets{#1}}
\let\yt@PopupPresets\@empty
```

⟨/pujs⟩
⟨∗package⟩

2.2 Search YouTube interactively

`\ytInputQuery[`⟨*opts*⟩`]{`⟨*wd*⟩`}{`⟨*ht*⟩`}` Provides a text field to enter a query string.

```
\newcommand{\ytInputQuery}[3][]{%
  \textField[\TU{Enter a query text string}#1]{ytSearchTxt}{#2}{#3}}
```

`\ytSearch[`⟨*opts*⟩`]{`⟨*wd*⟩`}{`⟨*ht*⟩`}` A push button what will search YouTube based on the query string. **Note:** `listType=search` is deprecated and will no longer be supported as of 15 November, 2020.

```
\newcommand{\ytSearch}[3][]{%
  \pushButton[\CA{Search}#1\AAmouseup{%
    var f=this.getField("ytSearchTxt");\r
    var v=f.value;\r
    if ( (v=v.replace(/\string\\s/g,"+")) != "" )\r\t
      app.launchURL("\ytURL/results?search_query="+v);
  }]{ytSearchBtn}{#2}{#3}}
```

`\ytClearQuery[`⟨*opts*⟩`]{`⟨*wd*⟩`}{`⟨*ht*⟩`}` A button to clear the query string.

```
\newcommand{\ytClearQuery}[3][]{%
  \pushButton[\CA{Clear}#1
    \AAmouseup{this.resetForm("ytSearchTxt");}
  ]{ytSearchClr}{#2}{#3}%
}
```

⟨/package⟩
⟨∗pujs⟩

2.3 Document JavaScript

Some JavaScript to process the user's choice and to launch the browser to view the selected video.

```
\begin{insDLJS*}{yt}
\begin{newsegment}{ltx4yt: %
Popup Menu Data and JavaScript support functions}
var YTdebug=false;
var aYTLastChoice=new Array;
var bYTLastChoice=false;
\ytPopupAllMenuData
var aChoice; // make local
function ytProcessMenu(cMenu) { // aMenu->cMenu now a string
  var aMenu=eval(cMenu);
  var cChoice = app.popUpMenuEx.apply( app, aMenu );
  ytProcessMenu.cChoice=cChoice;
  if ( cChoice != null ) {
    aChoice=eval(cChoice);
    if (aChoice[1]=="") return null;
    var thisChoice=aChoice[0];
    eval(cMenu+thisChoice).bMarked=true;
    if (!bYTLastChoice) {
      eval(cMenu+aChoice[0]).bMarked=true;
    } else {
  var structLoc=eval(aYTLastChoice[1])[0]
  eval(aYTLastChoice[0]+structLoc).bMarked=false;
      eval(cMenu+aChoice[0]).bMarked=true;
    }
    return aChoice;
  } else return null;
}
function ytPopupMenu(cMenu) { // cMenu now a string
  var aChoice=ytProcessMenu(cMenu);
  var cChoice=ytProcessMenu.cChoice;
  var aMenu=eval(cMenu);
  if (aChoice!=null) {
    var title=eval(cMenu+aChoice[0]).cName;
    var i=title.indexOf("*");
    var _hash=(i == -1)?"embed/"+aChoice[1]:"watch?v="+aChoice[1];
    if (!bYTLastChoice) {
      if(YTdebug) %
console.println("launching url https://www.youtube.com/"+_hash);
      else app.launchURL("https://www.youtube.com/"+_hash,false);
      aYTLastChoice=[cMenu,cChoice];
      bYTLastChoice=true;
    } else {
```

```
var cLastMenu=eval(aYTLastChoice[1])[0]
      aYTLastChoice=[cMenu,cChoice];
      if (cLastMenu!=aChoice[0]) {
        if (YTdebug) %
console.println("will launch url: https://www.youtube.com/"+_hash);
        else app.launchURL("https://www.youtube.com/"+_hash,false);
      } else {
        if (YTdebug) console.println("will NOT launch url");
        // choice is the same, uncheck this item
        eval(cMenu+aChoice[0]).bMarked=false;
        bYTLastChoice=false;
      }
    }
  }
}
\end{newsegment}
\end{insDLJS*}
```

⟨*pujs⟩
⟨*package⟩
\yt@restoreCats
⟨/package⟩

3 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.

4 Change History