The cooltooltips package*

Scott Pakin
scott+ctip@pakin.org

March 7, 2006

1 Introduction

The cooltooltips package enables a document to contain hyperlinks that pop up a brief tooltip when the mouse moves over them and also open a small window containing additional text. cooltooltips works only with pdfLaTeX. Furthermore, the tooltips that cooltooltips produces are much less cool when viewed under older versions of Acrobat (< 7.0) or the current version of xpdf (3.00) because they don't pop up the extra, small window. This text is an example of a cool tooltip (assuming you're viewing this document with a sufficiently capable PDF reader). Move your mouse pointer over it and watch what happens. Then, click on the link. If your PDF reader is properly configured it should launch a Web browser and send it to the CTAN home page.

If the cooltooltips popup mechanism causes problems with your browser you can click here to disable popups. (Click again to re-enable them.) Regardless of whether popups are enabled the tooltip and hyperlink mechanisms continue to function.

The cool tooltip shown above was created with the following code:

```
\cooltooltip
  [0 0 1]
  {Example}
  {This is an example of a cool tooltip. Pretty cool, eh?}
  {http://www.ctan.org/}{Visit CTAN on the Web}
  {This text\strut}
```

The "click here" button was created as follows:

```
\cooltooltiptoggle{\fcolorbox{blue}{white}{click here}}
```

*This document corresponds to cooltooltips v1.0, dated 2006/03/07.

2 Usage

`\cooltooltip [`⟨*popup color*⟩`] [`⟨*link color*⟩`]`
`{`⟨*subject*⟩`} {`⟨*message*⟩`} {`⟨URL⟩`} {`⟨*tooltip*⟩`} {`⟨*text*⟩`}`

The `\cooltooltip` macro takes two optional arguments and five mandatory arguments. The first argument, ⟨*popup color*⟩, is the color of the box containing the textual message to display and is specified as a "⟨*red*⟩ ⟨*green*⟩ ⟨*blue*⟩" triple with each element ranging from 0 (off) to 1 (on). If omitted, ⟨*popup color*⟩ defaults to "`0 1 0`" (bright green). The second argument, ⟨*link color*⟩, is the color of the frame drawn around the hyperlink. If omitted, it defaults to the same value as ⟨*popup color*⟩. ⟨*subject*⟩ is a text string to display as the subject of the popup window. ⟨*message*⟩ is a text string to display within the popup window. There's no provision for scrolling the popup window so ⟨*message*⟩ should be kept reasonably short. When a user clicks on the hyperlink, the PDF browser should take him to URL ⟨URL⟩. While the mouse is hovering over the link, the ⟨*tooltip*⟩ text is displayed. Finally, ⟨*text*⟩ is the text of the hyperlink and can be composed of arbitrary LaTeX text, including mathematics, graphics, etc.

The width of the hyperlink frame is governed by `\fboxrule` and the space separating the frame from ⟨*text*⟩ is governed by `\fboxsep`. Use LaTeX's `\setlength` command to assign values to those registers.

Figure 1 illustrates how Adobe Reader 7.0 displays ⟨*subject*⟩, ⟨*message*⟩, ⟨*tooltip*⟩, and ⟨*text*⟩ with a ⟨*popup color*⟩ of cyan (`0 1 1`) and a ⟨*link color*⟩ of magenta (`1 0 1`). (The URL specified by ⟨URL⟩ does not appear on screen.)

Because cooltooltips uses LaTeX's `\label`/`\pageref` mechanisms for accurately determining the current page, documents built using cooltooltips will need to be run through `pdflatex` at least twice. (`pdflatex` will issue the standard "`Rerun to get cross-references right`" message as a reminder.)

`\cooltooltiptoggle {`⟨*text*⟩`}`

The popup mechanism used by cooltooltips is extremely fragile. cooltooltips has to manually transfer focus among the hyperlink, popup, and a per-page invisible form field. (See Section 3.3 for details and an explanation of why this trickery is necessary.) If the browser window is so small that the popup overlaps the mouse pointer, the popup will flicker rapidly and impede the use of the hyperlink. Because this behavior is disturbing to readers of the document, the author may want to provide the reader with the ability to disable popups.

The `\cooltooltiptoggle` command converts its ⟨*text*⟩ argument to a toggle button. Pressing the button suppresses all popups in the document. Pressing it again re-enables popups.

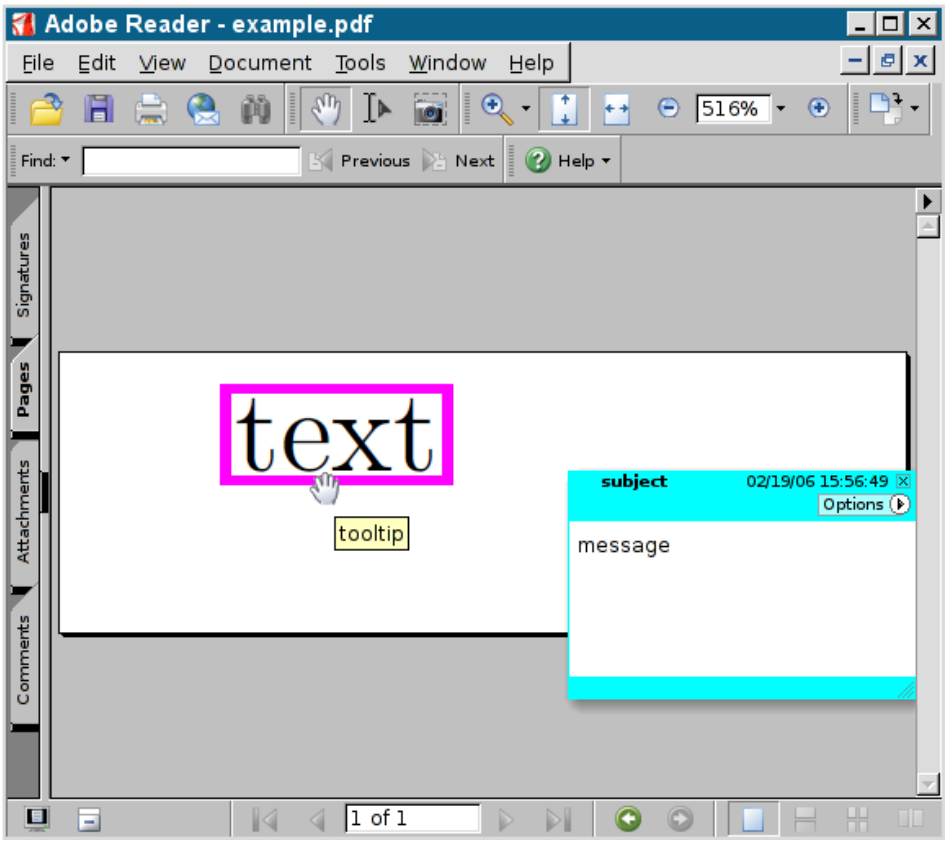

Figure 1: Illustration of `\cooltooltip` arguments

3 Implementation

This section presents the commented LaTeX 2_ε source code for the cooltooltips package. Each cool tooltip is implemented in terms of two PDF Annot objects. The popup is a Text annotation with an invisible appearance. The popup trigger/tooltip is both a Widget annotation and Btn pushbutton field. JavaScript code implements the popup open/close logic. For compatibility with PDF browsers that don't support Widget annotations we also include an ordinary Link annotation.

Section 3 is structured as follows. Section 3.1 marks the document as a PDF form, which is necessary for using fields and widgets. Section 3.2 defines macros for creating a Text annotation, cooltooltips's popup mechanism. Most of cooltooltips's behavior is defined in Section 3.3. All PDF fields/Widgets are specified in that section. The `\cooltooltip` command proper is defined in Section 3.4. Finally, Section 3.5 includes a tiny amount of extra code to verify that the document is being built under pdfLaTeX. If not, it disables all cooltooltips functionality but enables the document to build without it.

Because cooltooltips works only with pdfLaTeX and only in PDF mode, we load the ifpdf package up front to simplify testing for that case.

```
\RequirePackage{ifpdf}
```

3.1 AcroForm construction

PDF requires that all top-level form fields be pointed to by an AcroForm entry in the catalogue. We therefore have to keep track of all of our form fields.

`ctip@form@fields` Define a list of "⟨*object*⟩ `0 R`" elements.

```
\newcommand*{\ctip@form@fields}{}
```

At the end of the document we need to export the final value of `\ctip@form@fields` as an AcroForm.

```
\ifpdf
  \AtEndDocument{%
    \immediate\pdfobj {
      <<
        /Fields [\ctip@form@fields]
        /NeedAppearances true
      >>
    }%
    \pdfcatalog {
      /AcroForm \the\pdflastobj\space 0 R
    }%
  }
\fi
```

3.2 Text annotation construction

`\ctip@empty@icon` Define an empty XForm object to use as an invisible icon for the Text annotation.

```
\ifpdf
  \setbox\@tempboxa=\hbox{}
  \immediate\pdfxform\@tempboxa
  \edef\ctip@empty@icon{\the\pdflastxform}
\fi
```

`\ctip@tip@number` Keep track of the current tip number. This is necessary for generating unique object names.

```
\newcommand*{\ctip@tip@number}{0}
```

`\ctip@make@Text` Create a Text annotation with a given color (`#1`, optional), subject (`#2`), and content string (`#3`) and an invisible icon. This annotation is what will be popped up when the pointer enters the associated Widget.

```
\newcommand*{\ctip@make@Text}[3][0 1 0]{%
  \pdfannot width 0pt height 0pt depth 0pt {
    /Subtype /Text
```

```
    /C [#1]
    /Subj (#2)
    /Contents (#3)
    /NM (ctip Text \ctip@tip@number)
    /AP <<
      /N \ctip@empty@icon\space 0 R
      /D \ctip@empty@icon\space 0 R
      /R \ctip@empty@icon\space 0 R
    >>
    /Open false
  }%
}
```

3.3 Widget annotation construction

The Widgets in this section are also PDF pushbutton fields. PDF supports merging the two object dictionaries because their keys are disjoint.

`\ctip@current@page` Store the page number on which a Widget is finally placed.

```
\newcommand*{\ctip@current@page}{1}
```

`\ctip@last@invis` Keep track of the page number on which we last placed an invisible Widget.

```
\newcommand*{\ctip@last@invis}{0}
```

`\ctip@label` The amsmath package redefines `\label` within its equation environments. We need access to the original `\label` so we store a copy in `\ctip@label`.

```
\let\ctip@label=\label
```

`\ctip@update@pagenum` We can't reliably use `\thepage` to get the current page number (cf. `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=wrongpn`). Hence, we exploit the `\label`/`\pageref` mechanism to get an accurate page number. `\ctip@update@pagenum` creates a label (based on Section 3.2's `\ctip@tip@number`) then sets `\ctip@current@page` to the page on which the label occurs.

```
\newcommand*{\ctip@update@pagenum}{%
```

`\ctip@refname` Using `\label` to define a label ⟨*label name*⟩ implicitly defines a macro called `\r@`⟨*label name*⟩. That macro is `\relax` the first time `pdflatex` is run and thereafter expands to some number of values, the second of which is the label's page number.

```
  \ctip@label{ctip:tip:\ctip@tip@number}%
  \expandafter\let\expandafter\ctip@refname
    \csname r@ctip:tip:\ctip@tip@number\endcsname
  \@ifundefined{ctip@refname}{%
```

The first time through we use `\thepage` as an estimate for the correct page number.

```
    \xdef\ctip@current@page{\thepage}%
  }{%
```

On subsequent runs we extract the second (page number) argument and discard the rest.

```
    \def\ctip@secondofN##1##2##3!{%
      \xdef\ctip@current@page{##2}%
    }%
    \expandafter\ctip@secondofN\ctip@refname!%
  }%
}
```

`\ctip@make@invisible@Widget` For a Widget to return focus to the "background" it really returns focus to an *invisible* Widget. We need only one invisible Widget per page for this trick to work.

```
\newcommand*{\ctip@make@invisible@Widget}{%
  \pdfannot width 0pt height 0pt depth 0pt {
    /Subtype /Widget
    /FT /Btn
    /T (ctip invisible Widget \ctip@current@page)
    /DA (/Helv 10 Tf 0 0 0 rg)
    /Ff 65536
    /F 2
```

Sometimes, for various focusing trickery to work, the invisible Widget has to be made visible temporarily. We therefore define an action that makes the Widget invisible again as soon as it receives the input focus.

```
    /AA <<
      /Fo <<
        /Type /Action
        /S /JavaScript
        /JS (event.target.display = display.hidden)
      >>
    >>
  }%
}
```

`\ctip@content@box` The Widget's visual content is stored as a TEX box.

```
\newsavebox{\ctip@content@box}
```

`\ctip@unfocus@js` Although a PDF field can grab the input focus using the JavaScript `setFocus()` method, there's no mechanism in PDF 1.6 to explicitly reset the input focus to the page. The reason we want to clear the input focus is that the Page Up and Page Down keys function as expected only when none of the fields have the focus. `\ctip@unfocus@js` expands to some JavaScript code that implicitly resets the input focus. The trick is to make an invisible Widget temporarily visible, give it the focus, and let its focus (`Fo`) action make the Widget invisible again. Because an invisible Widget apparently can't retain the input focus, the focus is reset to the page.

```
\newcommand*{\ctip@unfocus@js}{%
  var ctipField =
```

```
    this.getField("ctip invisible Widget \ctip@current@page");
  ctipField.display = display.visible;
  ctipField.setFocus();
}
```

`\ctip@enter@js` The `\ctip@enter@js` macro expands to some JavaScript code to execute when the mouse pointer enters the (visible) Widget. The code instructs the associated Text annotation to open. If the global JavaScript variable `ctip_disable_popups` is set to `true` then `\ctip@enter@js` does nothing.

```
\newcommand*{\ctip@enter@js}{%
  if (!global.ctip_disable_popups) {
    var ctipText =
      this.getAnnot(this.pageNum, "ctip Text \ctip@tip@number");
    ctipText.popupOpen = true;
    \ctip@unfocus@js
  }
}
```

`\ctip@exit@js` The `\ctip@exit@js` macro expands to some JavaScript code to execute when the mouse pointer exits the (visible) Widget. The code instructs the associated Text annotation to close. The problem is that it doesn't close immediately but rather waits until focus leaves the Text annotation. (Opening the annotation apparently gives it focus.) Hence, we explicitly change the focus to the associated invisible Widget annotation to force the Text annotation to close immediately. Because the invisible Widget is invisible it can't steal the input focus from the page and thereby prevent the Page Up and Page Down keys from functioning properly. If the global JavaScript variable `ctip_disable_popups` is set to `true` then `\ctip@exit@js` does nothing.

```
\newcommand*{\ctip@exit@js}{%
  if (!global.ctip_disable_popups) {
    var ctipText =
      this.getAnnot(this.pageNum, "ctip Text \ctip@tip@number");
    ctipText.popupOpen = false;
    \ctip@unfocus@js
  }
}
```

`\ctip@make@Widget` Create a Widget annotation which represents a pushbutton field. `\ctip@make@Widget` expects `\ctip@content@box` to have the desired height, width, and depth of the Widget. The arguments to `\ctip@make@Widget` are the link color (`#1`, optional), the URL to link to (`#2`), the tooltip to display (`#3`).

```
\newcommand*{\ctip@make@Widget}[3][0 1 0]{%
```

Prepare to make the Widget annotation the same size as `\ctip@content@box` plus an `\fboxsep`+`\fboxrule`'s worth of space on each of its four sides.

```
  \setlength{\@tempdima}{\wd\ctip@content@box}%
  \addtolength{\@tempdima}{\fboxsep}%
  \setlength{\@tempdimb}{\ht\ctip@content@box}%
```

```
\addtolength{\@tempdimb}{0.5\fboxsep}%
\setlength{\@tempdimc}{\dp\ctip@content@box}%
\addtolength{\@tempdimc}{0.5\fboxsep}%
\hspace*{-0.5\fboxsep}%
```

`\ctip@action@object` Create a separate Action object because we intend to use it in both the Widget annotation and in a Link annotation. (See below.)

```
\immediate
\pdfobj {
  <<
    /Type /Action
    /S /URI
    /URI (#2)
  >>
}%
\edef\ctip@action@object{\the\pdflastobj\space 0 R}%
```

For compatibility with xpdf, which—as of this writing—does not support Widget annotations, we put a Link annotation behind the Widget annotation.

```
\makebox[0pt][l]{%
```

Because `\fboxrule` is rounded down when used as a Border width we (locally) increment it by ~ 1 to get it to round up instead.

```
  \advance\fboxrule by 0.9999pt
  \pdfannot width \@tempdima
            height \@tempdimb
            depth \@tempdimc {
    /Subtype /Link
    /A \ctip@action@object
    /Border [0 0 \strip@pt\fboxrule]
    /C [#1]
  }%
}%
```

We now create a Widget annotation, which is placed directly atop the Link annotation.

```
\pdfannot width \@tempdima
          height \@tempdimb
          depth \@tempdimc {
  /Subtype /Widget
  /FT /Btn
  /T (ctip Field \ctip@tip@number)
```

Acrobat 7.0, at least, displays the "alternate field name" (TU) as a tooltip.

```
  /TU (#3)
```

We're obligated to include a default appearance string (DA) even though we don't really need it here.

```
  /DA (/Helv 10 Tf 0 0 0 rg)
```

Set bit 17 (2^{17-1}) of the field flags (Ff) to indicate that this is a pushbutton—as opposed to a radio button or check box.

```
    /Ff 65536
```

Honor `\fboxrule` as the width of the link border.

```
    /BS <<
      /Type /Border
      /W \strip@pt\fboxrule
    >>
```

Define an appearance.

```
    /MK <<
      /BC [#1]
      /TP 1
    >>
```

Create an additional actions (AA) dictionary. This is where all of the popup magic is defined.

```
    /AA <<
```

When the mouse pointer enters the Widget we tell our associated Text annotation to open.

```
      /E <<
        /Type /Action
        /S /JavaScript
        /JS (\ctip@enter@js)
      >>
```

When the mouse pointer exits the Widget we tell our associated Text annotation to close.

```
      /X <<
        /Type /Action
        /S /JavaScript
        /JS (\ctip@exit@js)
      >>
```

When the user clicks on the Widget we relinquish the input focus and launch the specified URL.

```
      /U <<
        /Type /Action
        /S /JavaScript
        /JS (\ctip@unfocus@js)
        /Next \ctip@action@object
      >>
    >>
  }
```

Now that the Widget is defined we need to append an object reference for it to `\ctip@form@fields` so we can add that to the document's AcroForm.

```
  \xdef\ctip@form@fields{\ctip@form@fields\space\the\pdflastannot\space 0 R}
}
```

3.4 User commands

\cooltooltip The user can create a cool tooltip by invoking \cooltooltip with the popup color (#1, optional), the link color (#2, optional), the subject of the popup (#3), the string to display in the popup (#4), the URL to link to (#5), the tooltip to display (#6), and the text that will activate the tooltip/popup (#7).

```
\DeclareRobustCommand{\cooltooltip}[1][0 1 0]{%
  \def\ctip@popup@color{#1}%
  \ctip@cooltooltip@i
}
```

\ctip@cooltooltip@i This is where everything gets called. Upon entry, \ctip@popup@color is already defined as the popup color (and default color for the link).

```
\newcommand*{\ctip@cooltooltip@i}[6][\ctip@popup@color]{%
```

Store into \ctip@content@box the visual appearance of the link.

```
  \savebox{\ctip@content@box}{#6}%
```

Increase the cool tooltip number. \ctip@tip@number is used by \ctip@make@Widget, \ctip@make@Text, and \ctip@update@pagenum.

```
  \@tempcnta=\ctip@tip@number
  \advance\@tempcnta by 1
  \xdef\ctip@tip@number{\the\@tempcnta}%
```

Determine if we're on a new page and therefore need to create another invisible Widget.

```
  \ctip@update@pagenum
  \@tempcnta=\ctip@last@invis
  \@tempcntb=\ctip@current@page
  \ifnum\@tempcnta<\@tempcntb
    \ctip@make@invisible@Widget
    \xdef\c@ctip@last@invis{\ctip@current@page}%
  \fi
```

Place a Widget and its associated Text popup but without occupying any space on the page (as far as TEX can determine).

```
  \makebox[0pt][l]{%
    \ctip@make@Widget[#1]{#4}{#5}%
    \makebox[\paperwidth][r]{%
      \ctip@make@Text[\ctip@popup@color]{#2}{#3}%
    }%
  }%
```

Render the link contents that we stored earlier.

```
  \usebox{\ctip@content@box}%
}
```

\cooltooltiptoggle Cool tooltips are rather kludgy in the way that they manipulate PDF annotation state and transfer focus from annotation to annotation. Because this kludginess can lead to strange interactive behavior we provide the author with a \cooltooltiptoggle macro which creates a button to disable/enable the

cooltooltips popup mechanism. The sole argument to `\cooltooltiptoggle` is the button text.

```
\DeclareRobustCommand{\cooltooltiptoggle}[1]{%
  \savebox{\ctip@content@box}{#1}%
  \makebox[0pt][l]{%
    \pdfannot width \wd\ctip@content@box
              height \ht\ctip@content@box
              depth \dp\ctip@content@box {
      /Subtype /Link
      /Border [0 0 0]
      /A <<
        /Type /Action
        /S /JavaScript
        /JS (
          global.ctip_disable_popups = !global.ctip_disable_popups;
          var ctipField;
          var i;
          for (i=1; (ctipField=this.getField("ctip Field " + i)); i++)
            ctipField.display =
              global.ctip_disable_popups ? display.hidden : display.visible;
        )
      >>
    }%
  }%
  \usebox{\ctip@content@box}%
}
```

3.5 Sanity checks

Complain—but attempt to continue—if we're not running pdfLaTeX in PDF mode.

```
\RequirePackage{ifpdf}
\ifpdf
\else
  \PackageWarning{cooltooltips}{%
    Not running pdfLaTeX in PDF mode; disabling cooltooltips%
  }
  \renewcommand*{\ctip@cooltooltip@i}[6][]{\mbox{#6}}
\fi
```

4 Future work

There's unlikely to be any future work on cooltooltips; consider it to be a "dead" package. Yes, I know that someone will want a `dvipdfm` port and someone else will want finer-grained control over which of ⟨*subject*⟩, ⟨*message*⟩, ⟨URL⟩, and ⟨*tooltip*⟩ are utilized and a third person will request a less sloppy implementation of the code. However, I wrote cooltooltips primarily as a one-time-use package for my personal use; I needed a way to implement fancy popups for my Visual LaTeX FAQ

document and initially had no intention of distributing the popup mechanism as a separate package. If you find cooltooltips useful in its current form, that's great. If not, then you're on your own for fixing it; I don't plan on spending any significant time maintaining the package.

5 License agreement

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.