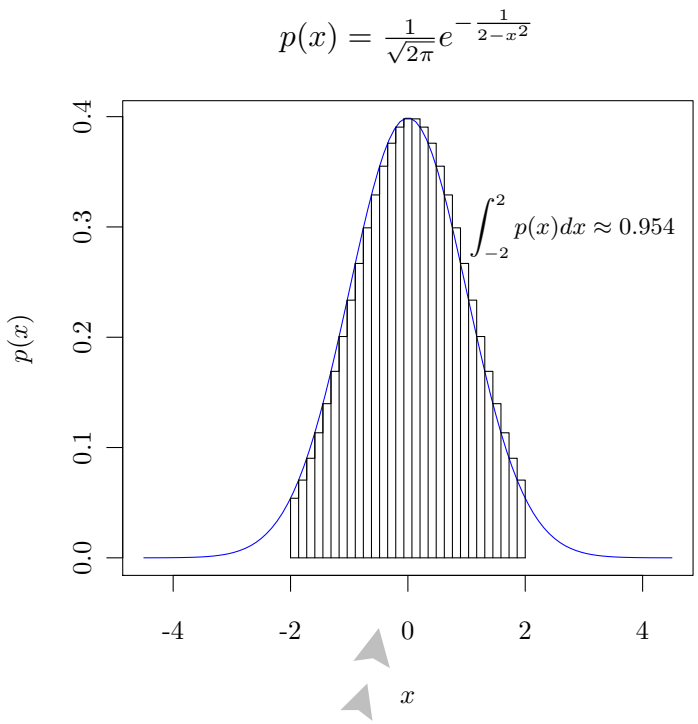

TikZ Device

LaTeX Graphics for R

The **tikzDevice** Package

http://r-forge.r-project.org/projects/tikzdevice

Charlie Sharpsteen and Cameron Bracken

Version 0.4-Beta July 23, 2009

Contents

Caveat Utilitor

This is a friendly reminder that the **tikzDevice** package is currently considered by it's designers to be a **beta work**. This package has been released in the hopes that it will help users produce exceptional graphics, leading to an increase in the quality of reports and other materials using those graphics.

However, the package has not been tested in an extensive variety of situations beyond those that have been dreamed up by the designers. Therefore, it is very likely that this package will help produce exceptional crashes, leading to an increase in the quantity of swearing by R and users affected by those crashes.

We hope this package will improve with time and attain a level of usability and trustability at which point it may be labeled "stable". Until then the package, including this documentation, may undergo significant changes in form and function. During the beta period the actual implementation of the package may differ significantly from what is described in this documentation.

In fact, the documentation may differ significantly from what is described in the documentation.

Thanks for giving it a try!
-The tikzDevice Team

1 Introduction

The **tikzDevice** package allows for R graphics output in a native LaTeX format. That is, the output of the `tikz()` function is plain-text files that can be interpreted using Ti*k*Z, a package for LaTeX. These files can be directly included in LaTeX documents by way of the `\input{}` statement. Allowing LaTeX to handle both typesetting and figure composition bestows the resulting document with a clean, unified look as there are no discontinuities in the size and selection of fonts used in the output.

This document is divided into three parts. The first part describes the functions that the package makes available to the R user and provides examples of their capabilities. Besides the R environment, use of the Ti*k*Z device device requires the user to have a working LaTeX compiler along with an installed version of the Ti*k*Z package. The second part of this documentation offers suggestions on how to get these tools working properly.

The third part of the documentation is intended for those who are curious as to the details of how this package has been implemented. It attempts to explain how the Ti*k*Z package does the things that it does and why it chooses to do them that way. The authors have attempted to write this part of the documentation in a way that is accessible to users as well as developers. This has been done in the hope that this project may serve as a case study in creating an R graphics device. This part of the documentation may also help those considering undertaking the transition from casual package-building to full-on hacking of the R internals.

Part I

Usage and Examples

2 Loading the Package

The functions in the **tikzDevice** package are made accessible in the R environment by using either the `library()` or `require()` functions like so:

```
require(tikzDevice)
```

Upon loading, the package will search for a usable LaTeX compiler. Access to LaTeX is essential for the device to produce correct output as the compiler is queried for font metrics several times during device output. For more information on why communication between the device and LaTeX is necessary, see Part III. If the search for a compiler is successful the package startup message should look similar to the following:

```
filehash: Simple key-value database (2.0-1 2008-12-19)
tikzDevice: A Device for R Graphics Output in PGF/TikZ Format (v0.3.5)
Checking for a LaTeX compiler...

pdfTeX 3.1415926-1.40.9-2.2 (Web2C 7.5.7)
kpathsea version 3.5.7
...

A working LaTeX compiler was found in:
        The R environment variable R_LATEXCMD

Global option tikzLatex set to:
        /usr/texbin/latex
```

If a working LaTeX compiler cannot be found, the **tikzDevice** package will fail to load and a warning message will be displayed:

```
An appropriate LaTeX compiler could not be found.
Access to LaTeX is currently required in order for the
TikZ device to produce output.

The following places were tested for a valid LaTeX compiler:

        A pre-existing value of the global option tikzLatex
        The R environment variable R_LATEXCMD
        The R environment variable R_PDFLATEXCMD
        The global option latexcmd
        The PATH using the command latex
        The PATH using the command pdflatex
...

Error : .onLoad failed in 'loadNamespace' for 'tikzDevice'
Error: package/namespace load failed for 'tikzDevice'
```

In this case, **tikzDevice** has done it's very best to locate a working compiler and came up empty. If you have a working LaTeX compiler, the next section describes how to inform the **tikzDevice** package of its location. For suggestions on how to obtain a LaTeX compiler, see Part II.

2.1 R Options That Affect Package Behavior

The **tikzDevice** package is currently influenced by a number of global options that may be set your R scripts, in the R console or in your in a `.Rprofile` file. All of the options can be set by using `options(<option> = <value>)`. These options allow for the use of custom `documentclass` declarations, LaTeX packages, and typesetting engines (e.g. XeLaTeX). The defaults , if are any for a given option, are shown below the description. The global options are:

tikzLatex Specifies the location of the LaTeX compiler to be used by **tikzDevice**. Setting a default for this this option may help the package locate a missing compiler:

Setting the default LaTeX compiler in `.Rprofile`

```
options( tikzLatex = '/path/to/latex/compiler' )
```

tikzMetricsDictionary When using the graphics device provided by **tikzDevice**, you may notice that R appears to "lag" or "hang" when commands such as `plot()` are executed. This is because the device must query the LaTeX compiler for string widths and font metrics. For a normal plot, this may happen dozens or hundreds of times- hence R becomes unresponsive for a while. The good news is that the `tikz()` code is designed to cache the results of these computations so they need only be performed once for each string or character. By default, these values are stored in a temporary cache file which is deleted when R is shut down. A location for a permanent cache file may be specified:

Setting a location in `.Rprofile` for a permanent metrics dictionary

```
options( tikzMetricsDictionary = '/path/to/dictionary/location' )
```

tikzDocumentDeclaration A string. The documentclass declaration when `standAlone == TRUE` as wel as when font metrics are calculated

Default

```
options( tikzDocumentDeclaration = "\\documentclass{article}" )
```

tikzFooter A character vector. The footer to be used only when `standAlone==TRUE`

Default

```
options( tikzFooter = c( "\\end{document}" ) )
```

tikzLatexPackages A character vector. These are the packages which are included when using the `standAlone` option as well as when font metric are calculated.

Default

```
options( tikzLatexPackages = c(
"\\usepackage{tikz}",
"\\usepackage[active,tightpage]{preview}",
"\\PreviewEnvironment{pgfpicture}",
"\\setlength\\PreviewBorder{0pt}") )
```

tikzMetricPackages A character vector. These are the extra packages which are additionally loaded when doing font metric calculations. As you see below, the font encoding is set to Type 1. This is very important so that character codes of LaTeX and R match up.

Default

```
options( tikzMetricPackages = c(
"\\usepackage[utf8]{inputenc}",
"\\usepackage[T1]{fontenc}",
"\\usetikzlibrary{calc}"))
```

For convenience the function `setTikzDefaults()` is provided which sets all the global options back to their original values.

The proper placement of a `.Rprofile` file is explained in `?Startup`. For the details of why calling the LaTeX compiler is necessary, see Part III.

A Word of Caution About Setting Options.

A lot of power is given to you through these global options, and with great power comes great responsibility. For example, if you do not include the Ti*k*Z package in the `tikzLatexPackages` option then all of the string metric calculations will fail. Or if you use a different font when compiling than you used for calculating metrics, strings may be placed incorrectly. There are innumerable ways for packages to clash in LaTeX so just be aware.

3 The `tikz()` Function

3.1 Description

The `tikz()` function provides most of the functionality of the **tikzDevice** package. This function is responsible for creating new R graphics devices that translate the output of graphics functions to the Ti*k*Z format. The device supports many levels of output that range from stand-alone LaTeX documents that may be compiled directly to code chunks that must be incorporated into existing LaTeX documents using the `\include{}` function.

3.2 Usage

The `tikz()` function opens a new graphics device and may be called with the following arguments:

```
tikz(file = "Rplots.tex", width = 7, height = 7,
    bg="white", fg="black", standAlone = FALSE, bareBones = FALSE,
    documentDeclaration = getOption("tikzDocumentDeclaration"),
    packages = getOption("tikzLatexPackages"),
    footer = getOption("tikzFooter")))
```

`file` A character string indicating the desired path to the output file. It is recommended, but not required, that the filename end in `.tex`.

`width` The width of the output figure, in **inches**.

`height` The height of the output figure, in **inches**.

`bg` The starting background color for the plot.

`fg` The starting foreground color for the plot.

standAlone A logical value indicating whether the resulting file should be suitable for direct processing by LaTeX.

bareBones A logical value indicating whether the resulting TikZcode produced without being placed within a LaTeX `tikzpicture` environment.

documentDeclaration See Section 2.1, "Options That Affect Package Behavior."

packages See Section 2.1, "Options That Affect Package Behavior."

footer See Section 2.1, "Options That Affect Package Behavior."

The first five options should be familiar to anyone who has used the default graphics devices shipped with R. The options `file`, `width`, `height`, `bg` and `fg` represent the standard graphics parameters currently implemented by **tikzDevice**. The last two options, `standAlone` and `bareBones`, are specific to the `tikz()` graphics device and affect the structure the output file. Using these options `tikz()` supports three modes of output:

- Graphics production as complete LaTeX files suitable for compilation.
- Graphics production as complete figures suitable for inclusion in LaTeX files.
- Graphics production as raw figure code suitable for inclusion in an enclosing `tikzpicture` environment in a LaTeX file.

The next section provides examples of how to use each type of output.

3.3 Examples

3.3.1 Default Mode

The most common use of the `tikz()` function is to produce a plot that will be included in another LaTeX document, such as a report. Running the following example in R will produce a very simple graphic using the `plot()` function.

```
require(tikzDevice)
tikz("simpleEx.tex", width = 3.5, height = 3.5)
plot(1, main = "Hello World!")
dev.off()
```

A simple LaTeX document is then required to display the output of the basic `tikz()` command. This document must include the TikZ as one of the packages that it loads. The TikZ package provides several optional libraries that provide additional functionality, however none of these libraries are currently required to use the output of `tikz()`. Inside the LaTeX document, the contents of the file simpleEx.tex are imported using the `\include{}` command.

Example LaTeX Document

```
\documentclass{article}

% All LaTeX documents including
% tikz() output must use this
% package!
\usepackage{tikz}
```

```
\begin{document}
  \begin{figure}[!h]
    \centering

    % The output from tikz()
    % is imported here.
    \input{simpleEx.tex}
```

```
    \caption{Simple Example}
  \end{figure}
\end{document}
```

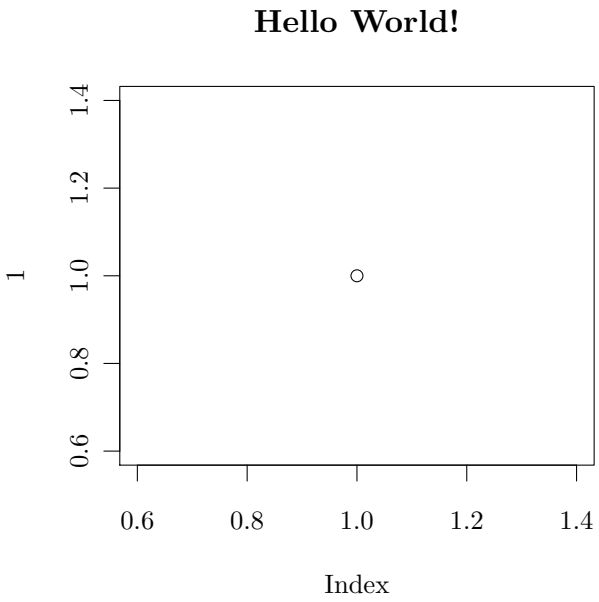

Figure 1: Example of simple `tikz()` usage.

One of the most exciting aspects of the `tikz()` function is that it allows the inclusion of arbitrary LaTeX code in plotting commands. An important issue to note is that many LaTeX commands pare prefixed by the backaslash, \, character. This character has a special meaning as an escape charcter in many computing applications, including R. Therefore, it is necessary to place two backslashes, \\, in the input to R commands in order to cause one to appear in the output. The next example demonstrates how to use LaTeX commands in plot annotation.

```
require(tikzDevice)
tikz('latexEx.tex',
  width=3.5,height=3.5)
x <- rnorm(10)
y <- x + rnorm(5,sd=0.25)
model <- lm(y ~ x)
rsq <- summary( model )$r.squared
rsq <- signif(rsq,4)
plot(x,y,main='Hello \\LaTeX!')
abline(model,col='red')
mtext(paste("Linear model: $R^{2}=",
  rsq,"$" ),line=0.5)
legend('bottomright', legend =
  paste("$y = ",
    round(coef(model)[2],3), 'x +',
    round(coef(model)[1],3), '$',
    sep=''), bty= 'n')
dev.off()
```

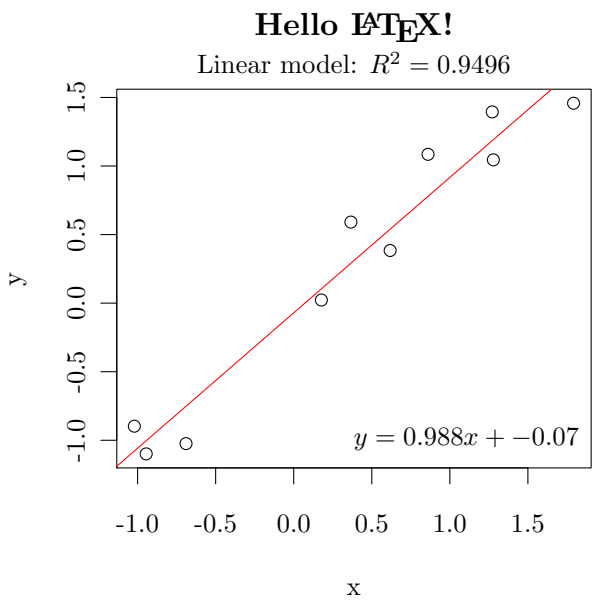

Figure 2: A more complicated example of `tikz()` usage incorporating natively rendered LaTeX commands.

3.3.2 `bareBones` Mode

The `barBones` is designed to facilitate inclusion of `tikz()` output as part of a larger graphic. Normally `tikz()` packages the commands it produces as a self-contained figure. This is done by placing the `\begin{tikzpicture}` and `\end{tikzpicture}` commands at the beginning and end of the output file. When `bareBones` is invoked, the `tikzpicture` environment is omitted which allows the output to be embedded inside a `tikzpicture` of the users own construction.

```
require(tikzDevice)
tikz('bareBonesExample.tex',width=4,height=4,bareBones=T)
x <- c(0,cumsum(rnorm(25)))
y <- c(0,cumsum(rnorm(25)))
plot(x, y, type="l" )
title("A Brownian Motion")
dev.off()
```

The resulting code may then be used inside a `tikzpicture` environment using the `\include{}` command. The included code must be wrapped in a `scope` environment that contains the options `x=1pt` and `y=1pt`. This informs Ti*k*Z of the units being used in the coordinates of the plot output. The options `xshift` and `yshift` may also be applied to the `scope` in order to position the plot. The following code demonstrates how to embed `bareBones` output in a `tikzpicture`

Example of a Ti*k*Z environment including `bareBones` output

```
\begin{tikzpicture}

  \draw[clip] (-0.25,-0.25) rectangle (4.25in,4.25in);

  \begin{scope}[x=1pt,y=1pt,yshift=0.25in]
    \input{figs/bareBonesExample}
  \end{scope}

  \node[anchor=south west,draw,rounded corners] (start) at (0,0)
    {TikZ can draw Brownian Motions as Well!};

  \coordinate (current point) at (start.east);
  \coordinate (old velocity) at (0,-5);
  \coordinate (new velocity) at (rand,rand);

  \foreach \i in {25,26,...,50} {

    \draw[blue!\i,ultra thick,line cap=round,x=1cm,y=1cm]
      (current point) .. controls ++([scale=-1]old velocity)
        and ++(new velocity) ..
      ++(rand, rand) coordinate(current point);

    \coordinate(old velocity) at (new velocity);
    \coordinate(new velocity) at (rand,rand);

  }

\end{tikzpicture}
```

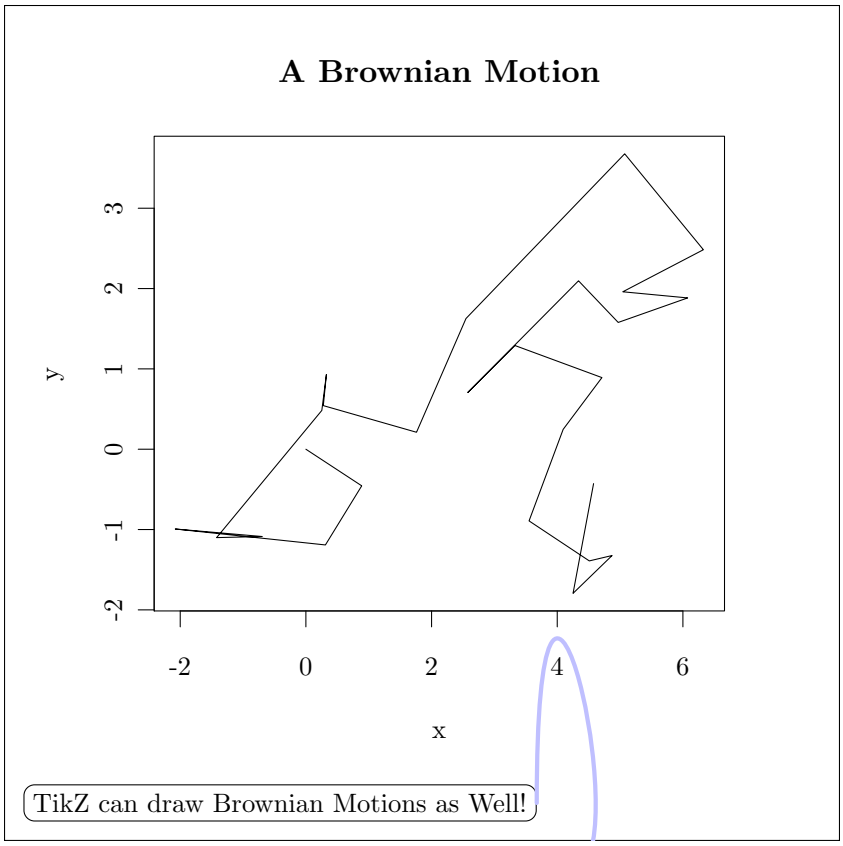

Figure 3: A Ti*k*Z drawing with embedded output from `tikz(bareBones=T)`.

3.3.3 `standAlone` Mode

When the `standAlone` option is passed to `tikz()`, the resulting `.tex` file will be a complete LaTeXdocument designed to be compiled on it's own. This means that in addition to `\begin{tikzpicture}` and `\end{tikzpicture}` the file will also contain `\begin{document}`, `\end{document}` and a LaTeX preamble. The `preview` package is also used in files produced by `standAlone` and is used to crop the pages in the resulting document to the bounding boxes of the figures that it contains. Stand-alone output may be produced in the following manner:

```
require(tikzDevice)
tikz('standAloneExample.tex',standAlone=T)
plot(sin,-pi,2*pi,main="A Stand Alone TikZ Plot")
dev.off()
```

Note that files produced using the `standAlone` option should not be included in LaTeXdocuments using the `\input{}` command! Use `\includegraphics{}` or load the `pdfpages` package and use `\includepdf{}`.

3.3.4 An XeLaTeX example

It is also possible to use XeLaTeX with **tikzDevice** that introduces some interesting possibilities. The following example was inspired by Dario Taraborelli and his article *The Beauty of LaTeX*.

XeLaTeX Example

```
#Set options for using XeLaTeX
options(tikzLatex = 'xelatex')
options(tikzDocumentDeclaration = '\\documentclass{article}')
options( tikzLatexPackages = c(
  "\\usepackage{fontspec}"
  ,"\\usepackage{tikz}"
  ,"\\usepackage{color}"
  ,"\\definecolor{Gray}{rgb}{.7,.7,.7}"
  ,"\\definecolor{lightblue}{rgb}{.2,.5,1}"
  ,"\\definecolor{myred}{rgb}{1,0,0}"
  ,"\\newcommand{\\red}[1]{\\color{myred} #1}"
  ,"\\newcommand{\\reda}[1]
   {\\color{myred}\\fontspec[Variant=2]{Zapfino}#1}"
  ,"\\newcommand{\\redb}[1]
   {\\color{myred}\\fontspec[Variant=3]{Zapfino}#1}"
  ,"\\newcommand{\\redc}[1]
   {\\color{myred}\\fontspec[Variant=4]{Zapfino}#1}"
  ,"\\newcommand{\\redd}[1]
   {\\color{myred}\\fontspec[Variant=5]{Zapfino}#1}"
  ,"\\newcommand{\\rede}[1]
   {\\color{myred}\\fontspec[Variant=6]{Zapfino}#1}"
  ,"\\newcommand{\\redf}[1]{\\color{myred}\\fontspec[Variant=7]{Zapfino}#1}"
  ,"\\newcommand{\\redg}[1]
   {\\color{myred}\\fontspec[Variant=8]{Zapfino}#1}"
  ,"\\newcommand{\\lbl}[1]{\\color{lightblue} #1}"
  ,"\\newcommand{\\lbla}[1]
   {\\color{lightblue}\\fontspec[Variant=2]{Zapfino}#1}"
  ,"\\newcommand{\\lblb}[1]
   {\\color{lightblue}\\fontspec[Variant=3]{Zapfino}#1}"
  ,"\\newcommand{\\lblc}[1]
   {\\color{lightblue}\\fontspec[Variant=4]{Zapfino}#1}"
  ,"\\newcommand{\\lbld}[1]
   {\\color{lightblue}\\fontspec[Variant=5]{Zapfino}#1}"
  ,"\\newcommand{\\lble}[1]
   {\\color{lightblue}\\fontspec[Variant=6]{Zapfino}#1}"
  ,"\\newcommand{\\lblf}[1]
   {\\color{lightblue}\\fontspec[Variant=7]{Zapfino}#1}"
  ,"\\newcommand{\\lblg}[1]
   {\\color{lightblue}\\fontspec[Variant=8]{Zapfino}#1}"
  ,"\\newcommand{\\old}[1]{"
  ,"\\fontspec[Ligatures={Common, Rare},%"
  ,"Variant=1,Swashes={LineInitial, LineFinal}]{Zapfino}"
  ,"\\fontsize{25pt}{30pt}\\selectfont #1}%"
  ,"\\newcommand{\\smallprint}[1]{\\fontspec{Hoefler Text}%"
  ,"\\fontsize{10pt}{13pt}\\color{Gray}\\selectfont #1}%"
))

#Set the content using custom defined commands
label <- c(
```

```
  "\\noindent{\\red d}roo{\\lbl g}"
  ,"\\noindent{\\reda d}roo{\\lbla g}"
  ,"\\noindent{\\redb d}roo{\\lblb g}"
  ,"\\noindent{\\redf d}roo{\\lblf g}\\\\[.3cm]"
  ,"\\noindent{\\redc d}roo{\\lblc g}"
  ,"\\noindent{\\redd d}roo{\\lbld g}"
  ,"\\noindent{\\rede d}roo{\\lble g}"
  ,"\\noindent{\\redg d}roo{\\lblg g}\\\\[.2cm]")

#Set the titles using custom defined commands, and hyperlings
title <- c(
    "\\smallprint{D. Taraborelli (2008),
      \\href{http://nitens.org/taraborelli/latex}%","{The Beauty of \\LaTeX}}"
    ,"\\smallprint{\\\\\\emph{Some rights reserved}.%
    ,"\\href{http://creativecommons.org/licenses/by-sa/3.0/}
      {\\textsc{cc-by-sa}}}"
)

#Draw the graphic
lim <- 0:(length(label)+1)
tikz('xelatexEx.tex',standAlone=T,width=5,height=5)
plot(lim,lim,cex=0,pch='.',xlab = 'Xe\\LaTeX{} Test',
    ylab='', main = title[1], sub = title[2])
for(i in 1:length(label))
    text(i,i,label[i])
dev.off()
```

Compiling the resulting file `xelatexEx.tex` like so:

Compileing with XeLaTeX

```
xelatex xelatexEx.tex
```

will produce the output in figure 4! Please note some of the fonts used in the example may not be available on your system.

4 The `getLatexCharMetrics()` and `getLatexStrWidth()` Functions

4.1 Description

These two functions may be used to retrieve font metrics through the interface provided by the **tikzDevice** package. Cached values of the metrics are returned if they have been calculated by the **tikzDevice** before. If no cached values exist, the LaTeX compiler will be invoked to generate them.

4.2 Usage

The font metric functions are called as follows:

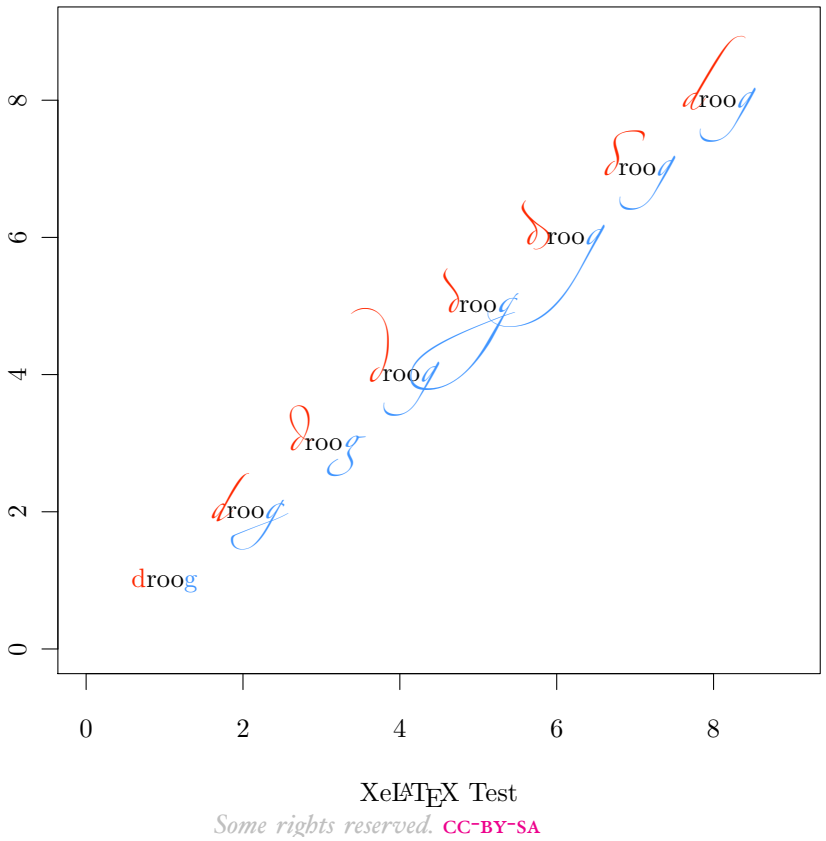

Figure 4: Result of XeLaTeX example

```
getLatexStrWidth( texString, cex = 1, face= 1)

getLatexCharMetrics( charCode, cex = 1, face = 1 )
```

texString A string for which to compute the width. LaTeX commands may be used in the string, however all backslashes will need to be doubled.

charCode An integer between 32 and 126 which indicates a printable character in the ASCII symbol table using the T1 font encoding.

cex The character expansion factor to be used when determining metrics.

face An integer specifying the R font face to use during metric calculations. The accepted values are as follows:

1: Text should be set in normal font face.

2: Text should be set in **bold font face**.

3: Text should be set in *italic font face*.

4: Text should be set in ***bold italic font face***.

5: Text should be interpreted as `plotmath` symbol characters. Requests for font face 5 are currently ignored.

4.3 Examples

The `getLatexStrWidth()` function may be used to calculate the width of strings containing fairly arbitrary LaTeX commands. For example, consider the following calculations:

```
getLatexStrWidth("The symbol: alpha")
[1] 82.5354

getLatexStrWidth("The symbol: $\\alpha$")
[1] 65.08636
```

For the first calculation, the word "alpha" was interpreted as just a word and the widths of the characters 'a', 'l', 'p', 'h' and 'a' were included in the string width. For the second string, `\alpha` was interpreted as a mathematical symbol and only the width of the symbol 'α' was included in the string width.

The `getLatexCharWidth()` function must be passed an integer corresponding to an ASCII character code and returns three values:

- The `ascent` of the character- the distance between the baseline and the highest point of the character's glyph.
- The `descent` of the character- the distance between the baseline and the lowest point of the character's glyph.
- The width of the character.

The character 'y' has an `ASCII` symbol code of 121 and possesses a tail that descends below the text line. Therefore a non-zero value will be returned for the descent of 'y'. The character 'x', `ASCII` code 120, has no descenders, so its descent will be returned as zero.

```
# Get metrics for 'y'
getLatexCharMetrics(121)
[1] 4.30450 1.94397 5.27649

# Get metrics for 'x' - the second value is the descent
# and should be zero or very close to zero.
getLatexCharMetrics(120)
[1] 4.30450 0.00000 5.27649
```

Note that characters, along with numbers outside the range of [32-126], may not be passed to the `getLatexCharMetrics()` function. If for some reason a floating point number is passed, it will be floored through conversion by `as.integer()`.

```
getLatexCharMetrics('y')
NULL

getLatexCharMetrics(20)
NULL

# Will return metrics for 'y'
getLatexCharMetrics(121.99)
[1] 4.30450 1.94397 5.27649
```

Part II
Installation Guide

This section is intended to offer pointers on how to obtain a LaTeX distribution if there is not one already installed on your system. The distributions detailed in this section are favorites of the **tikzDevice** developers as they integrated package managers which greatly simplify the process of installing additional LaTeX packages. Currently this section is not, and may never be, a troubleshooting guide for LaTeX installation. For those unfortunate situations we refer the user to the documentation of each distribution.

5 Obtaining a LaTeX Distribution

A LaTeX distribution provides the packages and support programs required by the **tikzDevice** and the documents that use its output. In addition a LaTeX compiler, a few extension packaged are required. Section 6 describes how to obtain and install these packages.

5.1 Windows

Windows users will probably prefer the MiKTeX distribution available at `http://www.miktex.org`. An amazing feature of the MiKTeX distribution is that it contains a package manager that will attempt to install missing packages on-the-fly. Normally when LaTeX is compiling a document that tries to load a missing package it will wipe out with a warning message. When the MiKTeX compilers are used compilation will be suspended while the new package is downloaded.

5.2 UNIX/Linux

For users running a Linux or UNIX operating system, we recommend the TeX Live distribution which is available at `http://www.tug.org/texlive/acquire.html`. TeX Live is maintained by the TeX Users Group and a new version is released every year. Note that the version of TeX Live provided by many Linux package management systems is the 2007 version. We recommend using TeX Live 2008 or higher as the `tlmgr` package manager was introduced in the 2008 distribution. Using `tlmgr` greatly simplifies the adding and removing packages from the distribution. The website offers an installation package, called `install-tl.tar.gz` or something similar, that contains a shell script that can be used to install an up-to-date version of the TeX Live distribution.

5.3 Mac OS X

For users running Apple's OS X, we recommend the Mac TeX package available at `http://www.tug.org/mactex/`. Mac TeX is basically TeX Live packaged inside a convenient OS X installer along with a few add-on packages. One striking difference between the Mac TeX and TeX Live installers is that the installer for Mac TeX includes the whole TeX Live distribution in the initial download- for TeX Live 2008 this amounts to approximately 1.2 GB. This is quite a large download that contains several packages that the average or even advanced user will never ever use. To conserve time and space we recommend installing from the basic installer at `http://www.tug.org/mactex/morepackages.html` and using the `tlmgr` utility to add desired add-on packages.

Adam R. Maxwell has created a very nice graphical interface to `tlmgr` for OS X called the TeX Live Utility. It may be obtained from `http://code.google.com/p/mactlmgr/` and we highly recommend it.

6 Installing Ti*k*Z and Other Packages

Unsurprisingly, **tikzDevice** requires the Ti*k*Z package to be installed and available in order to function properly. Ti*k*Z is an abstraction of a lower-level graphics language called PGF and both are distributed as the the `pgf` package.

6.1 Using a LaTeX Package Manager

The easiest way to install LaTeX packages is by using a distribution that includes a package manager such as MiKTeX or TeX Live/Mac TeX. For Windows users, the MiKTeX package manager usually handles package installation automagically during compilation of a document that is requesting a missing package. The MiKTeX package manager, `mpm`, can also be run manually from the command prompt:

Using `mpm` to install packages

```
mpm --install packagename
```

For versions of TeX Live and Mac TeX dated 2008 or newer, the `tlmgr` package manager is used in an almost identical manner:

Using `tlmgr` to install packages

```
tlmgr install packagename
```

6.2 Manual Installation

Sometimes an automated package manager cannot be used. Common reasons may be that one is not available, as is the case with the TeX Live 2007 distribution, or that when running the package manager you do not have write access to the location where LaTeX packages are stored, as is the case with accounts on shared computers. If this is the case, a manual install may be the best option for making a LaTeX package available.

Generally, the best place to find LaTeXpackages is the Comprehensive TeX Archive Network, or CTAN located at http://www.ctan.org. In the case of the PGF/Ti*k*Z package, the project homepage at http://www.sourceforge.net/projects/pgf is also a good place to obtain the package- especially if you would like to play with the bleeding-edge development version.

Generally speaking, all LaTeX packages are stored in a specially directory called a `texmf` folder. Most TeX distributions allow for each user to have their own personal `texmf` folder somewhere in their home path. The most usual locations, and here ***usual*** is and unfortunately loose term, are as follows:

For UNIX/Linux

```
~/texmf
```

For Mac OS X

```
~/Library/texmf
```

For Windows, using MiKTeX

```
# None predefined. However the following command will open
# the MiKTeX options panel and a new texmf folder may be assigned
# under the "Roots" tab.
mo
```

The location of files and subfolders in the `texmf` directory should follow a standard pattern called the TEX Directory Structure or TDS which is documented here: http://tug.org/tds/tds.pdf. Fortunately, most packages available on CTAN are archived in such a way that they will unpack into a TDS-complient configuration. TDS-complient archives usually have the phrase `tds` somewhere in their filename and may be installed from a UNIX shell[1] like so:

Installing LaTeX package archives

```
# For zip files.
unzip package.tds.zip -d /path/to/texmf

# For tarballs.
tar -xzf -C /path/to/texmf package.tar.gz
```

For packages that aren't provided in TDS-complient form look for installation notes- usually provided in the form of an `INSTALL` file. If all else fails LaTeX packages can usually be installed by copying the files ending in `.sty` to `texmf/tex/latex/`.

After package files have been unpacked to a `texmf` folder, the database of installed packages needs to be updated for the LaTeX compiler to take notice of the additions. This is done with the `mktexlsr` command:

Registering new LaTeX packages

```
mktexlsr

# Successful package installation can be checked by running the
# kpsewhich command. For a package accessed in a document
# by \usepackage{package}, kpsewhich should return a path to
# package.sty
kpsewhich tikz.sty
/Users/Smithe/Library/texmf/tex/latex/pgf/frontendlayer/tikz.sty
```

[1] Sorry Windows users, we enjoy using command prompt about as much as a poke in the eye with a sharp stick. Hence we don't use it enough to offer advice. May we suggest Cygwin?

Part III
Package Internals

7 Background

About Ti*k*Z, the PicTEX device and the story of two fortran programmers with a dream.

8 System Requirements

9 Character Metrics and String Width

9.1 Dictionaries

9.2 What the Heck are Acent, Decent and Width

10 On the Importance of Font and Style Consistancy in Reports

If you haven't figured it out by now, we are quite picky about the way our graphics look and the way

11 The pgfSweave Package and Automatic Report Generation

Now for a little shameless self promotion. The authors of **tikzDevice** have another package called **pgfSweave** which provides a driver for Sweave. **pgfSweave** started as an interface to eps2pgf and its ability to interpret strings in eps files as LaTeX. This was used to much the same effect as **tikzDevice**. The problem was the conversion from eps to pgf was SLOW. Long story short, by combining this functionality with the externalization feature of pgf and the **cacheSweave** we were able to achieve bearable compilation speed and nice looking graphics. **pgfSweave** is in the process of getting pumped up by interfacing with the **tikzDevice** package. We hope that the combination will be a self-caching, consistancy-enducing, user-empowering tool for high quality reports.