Essentials

- Tk Backgrounder (../resources/backgrounder.html)
- Installing Tk (../tutorial/install.html)
- Tutorial (../tutorial/index.html)
- Widget Roundup (../widgets/index.html)
- Languages Using Tk (../resources/languages.html)
- Official Tk Command Reference
  (Tcl-oriented; at www.tcl.tk) (http://www.tcl.tk/man/tcl8.6/TkCmd/contents.htm)

Tutorial

Show: All Languages ▾

This tutorial will quickly get you up and running with the latest Tk from Tcl, Ruby, Perl or Python on Mac, Windows or Linux. It provides all the essentials about core Tk concepts, the various widgets, layout, events and more that you need for your application.

Previous: Organizing Complex Interfaces (complex.html) | Contents (index.html) | Single Page (onepage.html) | Next: Canvas (canvas.html)

Fonts, Colors, Images

This chapter describes how Tk handles fonts, colors and images. We've touched on all of these before, but here we'll provide a more in depth treatment.

Fonts

Several Tk widgets, such as the label, text, and canvas, allow you to specify the fonts used to display text, typically via a `"font"` configuration option. As with many things in Tk, the default fonts are usually a good choice, but if you do want to make changes, this section will describe several ways to do so. Fonts are one of several areas that are highly platform-specific, so how you specify them is important.

The font command reference (http://www.tcl.tk/man/tcl8.6/TkCmd/font.htm) provides full details on specifying fonts, as well as other font-related operations.

Most of the themed widgets that display text don't have a `"font"` configuration option, unlike the classic Tk widgets. Rather than modifying individual widgets, the correct approach in the themed widgets is to specify the fonts used in a style, and then use that style for the individual widget. This is akin to the difference between hardcoding display-oriented markup like font tags inside HTML pages, vs. using CSS stylesheets that keep all the display specific information in one place.

> *Many older Tk programs hardcoded a lot of fonts, using either the "family size style" format we'll see below, X11 font names, or the older and more arcane X11 font specification string. In many cases, this left these applications with a dated look as platforms evolved. Further, many programs specified fonts on a per-widget basis, leaving the font decisions spread out through the program. Named fonts, and use of the standard fonts that Tk provides are a far better solution. Reviewing and updating the usage of fonts is an easy and important change to make in any existing applications.*

Standard Fonts

Particularly for more-or-less standard user interface elements, each platform defines specific fonts that should be used. Tk encapsulates many of these into a standard set of fonts that are always available, and, of course, the standard widgets use these fonts. This helps abstract away platform differences. The predefined fonts are:

`TkDefaultFont`	Default for items not otherwise specified.
`TkTextFont`	Used for entry widgets, listboxes, etc.
`TkFixedFont`	A standard fixed-width font.
`TkMenuFont`	The font used for menu items.
`TkHeadingFont`	Font for column headings in lists and tables.
`TkCaptionFont`	A font for window and dialog caption bars.
`TkSmallCaptionFont`	A smaller caption font for tool dialogs.
`TkIconFont`	A font for icon captions.
`TkTooltipFont`	A font for tooltips.

Platform-Specific Fonts

A number of additional predefined fonts are available, but the precise set depends on the platform. Obviously, if you're using these, and your application is portable across different platforms, you'll need to ensure that proper fonts are defined individually for each platform.

On X11, any valid X11 font name (see e.g. the "xlsfonts" command) may be used. Remember though that there is no guarantee that a particular font has been installed on a particular machine.

On Windows, the following font names, which map to the fonts that can be set in the "Display" Control Panel, are available: `system`, `ansi`, `device`, `systemfixed`, `ansifixed`, `oemfixed`.

On macOS, the following fonts are available (see the Apple HIG for details): `systemSystemFont`, `systemSmallSystemFont`, `systemApplicationFont`, `systemViewsFont`, `systemMenuItemFont`, `systemMenuItemCmdKeyFont`, `systemPushButtonFont`, `systemAlertHeaderFont`, `systemMiniSystemFont`, `systemDetailEmphasizedSystemFont`, `systemEmphasizedSystemFont`, `systemSmallEmphasizedSystemFont`, `systemLabelFont`, `systemMenuTitleFont`, `systemMenuItemMarkFont`, `systemWindowTitleFont`, `systemUtilityWindowTitleFont`, `systemToolbarFont`, `systemDetailSystemFont`.

> *If you'd like to examine what actual font is being assigned to any of the standard or default fonts (or any font for that matter), you can use the "font actual" call.*

Named Fonts

You can also create your own fonts, which can be used exactly like the predefined ones. To do so, you'll need to pick a name to refer to the font, and then specify various font attributes that define how the font should look. Typically, you'd use different font attributes on different platforms; that way, you can use the font in your program without worrying about the details except in the one place the font is actually defined.

Here's an example:

```
font create AppHighlightFont -family Helvetica -size 12 -weight bold
grid [ttk::label .l -text "Attention!" -font AppHighlightFont]
```

```
AppHighlightFont = TkFont.new :family => 'Helvetica', :size => 12, :weight => 'bold'
Tk::Tile::Label.new(root) {text 'Attention!'; font AppHighlightFont}.grid
```

```
Tkx::font_create("AppHighlightFont", -family => "Helvetica", -size => 12, -weight => "bold");
$l = $parent->new_ttk__label(-text => "Attention!", -font => "AppHighlightFont");
$l->g_grid;
```

```
from tkinter import font
appHighlightFont = font.Font(family='Helvetica', size=12, weight='bold')
ttk.Label(root, text='Attention!', font=appHighlightFont).grid()
```

The "`family`" specifies the font name; the names `Courier`, `Times`, and `Helvetica` are guaranteed to be supported (and mapped to an appropriate monospaced, serif, or sans-serif font), but other fonts installed on the system can be used (again, be careful to ensure the font exists, or the system will supply a different font, which may not necessarily be a good match). You can get the names of all available fonts with:

```
font families
```

```
TkFont.families
```

```
Tkx::font_families
```

```
font.families()
```

The "`size`" option specifies the size of the font, in points. The "`weight`" option can be either `bold` or `normal`. You can specify a "`slant`" of `roman` (normal) or `italic`. Finally, the boolean options "`underline`" and "`overstrike`" are available.

The current settings of these options can be examined or changed using the same mechanisms that you'd use for changing the configuration options of a widget (e.g. `configure`).

Font Descriptions

Another way to specify fonts is via a list of attributes, starting with the name of the font, and then optionally including a size, and optionally one or more style options. Some examples of this are `"Helvetica"`, "Helvetica 12", `"Helvetica 12 bold"`, and `"Helvetica 12 bold italic"`. These font descriptions are then used as the value of the `"font"` configuration option, rather than a predefined or named font.

> *In general, switching from font descriptions to named fonts is advisable, again to isolate font differences in one location in the program.*

Colors

As with fonts, there are various ways to specify colors. Full details can be found in the colors command reference (http://www.tcl.tk/man/tcl8.6/TkCmd/colors.htm).

In general, the system will provide the right colors for most things. Like with fonts, both Mac and Windows specify a large number of system-specific color names (see the reference), whose actual color may depend upon system settings (e.g. text highlight colors, default backgrounds).

You can also specify colors via RGB, like in HTML, e.g. `"#3FF"` or `"#FF016A"`. Finally, Tk recognizes the set of color names defined by X11; normally these are not used, except for very common ones such as `"red"`, `"black"`, etc.

For themed Tk widgets, colors are often used in defining styles that are applied to widgets, rather than applying the color to a widget directly.

It probably goes without saying that restraint in the use of colors is normally warranted.

Images

We've seen the basics of how to use images already, displaying them in labels or buttons for example. We create an image object, usually from a file on disk.

```
image create photo imgobj -file "myimage.gif"
.label configure -image imgobj
```

```
image = TkPhotoImage.new(:file => "myimage.gif")
label['image'] = image
```

```
Tkx::image_create_photo("imgobj", -file => "myimage.gif");
$l->configure(-image => "imgobj");
```

```
imgobj = PhotoImage(file='myimage.gif')
label['image'] = imgobj
```

Out of the box, Tk includes support for GIF and PPM/PNM images. Tk 8.6 added PNG to this short list. However, there is a Tk extension library called "Img" which adds support for many others: BMP, XBM, XPM, PNG, JPEG, TIFF, etc. Though not included directly in the Tk core, Img is usually included with other packaged distributions (e.g. ActiveTcl).

```
package require Img
image create photo myimg -file "myimage.png"
```

```
require 'tkextlib/tkimg'
myimg = TkPhotoImage.new(:file => 'myimage.png')
```

```
Tkx::package_require("Img");
Tkx::image_create_photo("myimg", -file => "myimage.png");
```

Instead of using Tk's Img extension, Tkinter uses a made-for-Python image library called 'PIL' (Python Imaging Library). More specifically, we'll use a more up-to-date fork of PIL called 'pillow'. As it doesn't come bundled with Python, you'll normally need to install it. You should be able to do so via, e.g. '`pip install Pillow`'.

```
from PIL import ImageTk, Image
myimg = ImageTk.PhotoImage(Image.open('myimage.png'))
```

The 'ImageTk.PhotoImage' call provides a drop-in replacement for Tk's PhotoImage, but supports the broader range of image types.

Tk's images are actually quite powerful and sophisticated and provide a wide variety of ways to inspect and modify images. You can find out more from the image command reference (http://www.tcl.tk/man/tcl8.6/TkCmd/image.htm) and the photo command reference (http://www.tcl.tk/man/tcl8.6/TkCmd/photo.htm).

> *The types of multi-color images we've seen here are referred to in Tk as photo images. Tk also provides a second type of images, two-bit bitmap images, which were widely used in the 90's when most Unix workstations used quite large (compared with PCs) monitors, but they were only black and white. Needless to say, color is mostly de rigueur these days, so updating to full-color images for icons and so on is highly advisable. Though in what some may consider retro-styling, some flat and material design schemes have returned to black and white. Plus ça change...*

Problems with PIL/Pillow on macOS?

If you're running on macOS, did you get a crash using PIL/Pillow? Are you maybe seeing error messages in your Terminal saying things like:

```
Class TKApplication is implemented in both
/Library/Frameworks/Tk.framework/Versions/8.5/Tk and
/System/Library/Frameworks/Tk.framework/Versions/8.5/Tk.
One of the two will be used. Which one is undefined.
```

This can occur when you download a binary of an extension which is linked against a different version of Tcl and Tk than you're running. In this case, you're running the ActiveTcl one in /Library/Frameworks, and the extension is trying to link against the one provided by macOS in /System/Library/Frameworks.

This is one of the difficulties with binary extensions. The snippet below shows how we might fix this problem. It's specific to the version of the Pillow extension that we happened to have installed, but hopefully it can guide you in terms of locating and fixing the binaries for any extension causing problems for you. The steps are roughly: find the binary, verify it is linking against the wrong version (via `otool`), and change what it's linking against, using a program called `install_name_tool`.

```
% cd /Library/Frameworks/Python.framework/Versions/3.4
% find . -name '_imagingtk.so'
./lib/python3.4/site-packages/PIL/_imagingtk.so
% cd lib/python3.4/site-packages/PIL
% otool -L _imagingtk.so
_imagingtk.so:
        /System/Library/Frameworks/Tcl.framework/Versions/8.5/Tcl (...)
        /System/Library/Frameworks/Tk.framework/Versions/8.5/Tk (...)
        /usr/lib/libSystem.B.dylib (...)
% install_name_tool -change /System/Library/Frameworks/Tcl.framework/Versions/8.5/Tcl
        /Library/Frameworks/Tcl.framework/Versions/8.5/Tcl _imagingtk.so
% install_name_tool -change /System/Library/Frameworks/Tk.framework/Versions/8.5/Tk
        /Library/Frameworks/Tk.framework/Versions/8.5/Tk _imagingtk.so
% otool -L _imagingtk.so
_imagingtk.so:
        /Library/Frameworks/Tcl.framework/Versions/8.5/Tcl (...)
        /Library/Frameworks/Tk.framework/Versions/8.5/Tk (...)
        /usr/lib/libSystem.B.dylib (...)
```

Previous: Organizing Complex Interfaces (complex.html) | Contents (index.html) | Single Page (onepage.html) | Next: Canvas (canvas.html)