On Scaling Support in Tk 8.7

by

Csaba Nemethi

csaba.nemethi@t-online.de

Contents

---

1. Abstract

Since nowadays more and more displays have a high pixel density (full HD or even 4K resolution paired with a relatively small size, like 14" or less), it is becoming increasingly important to develop the applications in such a way that the sizes of the various GUI components will automatically adapt themselves to the display's DPI scaling level.

This talk is about:

- my work on scalability in connection with Tablelist and Scrollutil;
- the new variables **`tk::scalingPct`** and **`tk::svgFmt`**, and how they are set at Tk 8.7 initialization time;
- how the built-in SVG support in Tk 8.7 has made it possible to make several Tk and Ttk widgets as well as the images used in the standard dialogs scaling-aware;
- the additional changes in the C code and library scripts that made the built-in Tk and Ttk widgets as well as the standard dialogs scaling-aware in Tk 8.7;
- how to make sure that the sizes of the various GUI components of newly developed Tk applications will automatically adapt themselves to the display's DPI scaling level;
- a few tips for transforming legacy Tk applications into scaling-aware ones;
- a case study: the steps that made the **Widget Demonstration** application scaling-aware in Tk 8.7.

---

2. The scaleutil Package

Large parts of the implementation of scaling support in Tk 8.7 are based on code contained in the **scaleutil** package.

- Work started in 2020 in connection with the Tablelist package. Currently at version 1.11.
- Bundled with Tablelist and Scrollutil, with comprehensive documentation in **Tablelist Programmer's Guide** and **Scrollutil Programmer's Guide**.
- Main public procedure **`scaleutil::scalingPercentage`**, invoked by **`package require tablelist(_tile)`** and **`package require scrollutil(_tile)`**, returns one of the scaling percentage values `100`, `125`, `150`, `175`, or `200`, after scaling a series of Tk core and Ttk widgets according to the *real* scaling percentage, which is held in the variable **`scaleutil::scalingPct`** and can be greater than `200`. The return value is assigned to the variable **`tablelist::scalingpct`** and **`scrollutil::scalingpct`**, respectively.

---

3. Two New Tk Variables

Tk 8.7 introduces the new variables **`tk::scalingPct`** and **`tk::svgFmt`**, both documented in the **tkvars** manual page:

- Tk sets the variable **`tk::scalingPct`** at initialization time to the scaling percentage corresponding to the display's DPI scaling level. This value is at least `100` and is restricted to multiples of `25`. The sizes and various attributes of the Tk core and Ttk widgets and their components, as well as the sizes of the images used by Tk are chosen according to the scaling percentage, and this is recommended for applications and library packages, too.

  On the windowing systems **`win32`** and **`aqua`** the scaling percentage is *basically* **`[expr {[tk scaling] * 75}]`**. On **`x11`** it is computed mostly (but not exclusively) from the value of the X resource **`Xft.dpi`**, and, as an additional step, Tk synchronizes the scaling factor used to convert between physical units and pixels with the scaling percentage, by invoking

  ```
  tk scaling [expr {$tk::scalingPct / 75.0}]
  ```

- The variable **`tk::svgFmt`** is set at Tk initialization time to

  ```
  [list svg -scale [expr {$tk::scalingPct / 100.0}]]
  ```

  It is recommended to pass the value of this variable to the commands **`image create photo`**, *`imageName`* **`configure`**, *`imageName`* **`put`**, and *`imageName`* **`read`** as the value of their **`-format`** option when creating or manipulating SVG images.

---

4. The Implementation of Scaling Support in Tk 8.7

The scaling support in Tk 8.7 is the result of the following steps:

- Added support for SVG images (back in 2018, thanks to **René Zaumseil**), based on the **tksvg** extension by **Christian Gollwitzer** (thanks also to **Harald Oehlmann** and **Christian Werner** for their contributions).

- On X11, replaced the sizes of the standard fonts in pixels with sizes in points (back in February 2020, for Tk 8.7a5 and later, thanks to **François Vogel**).

- Fixed a long-standing bug in the implementation of the indicators of the ttk::checkbutton and ttk::radiobutton widgets of the **`vista`** and **`xpnative`** themes, thus making sure that they will appear properly scaled (back in May 2020, for Tk 8.6.11 and later and 8.7a5 and later).

- Added the variables **`tk::scalingPct`** and **`tk::svgFmt`**, as described in the previous section.

- Replaced a series of legacy GIF images in the Tk library files with SVG icons, created using the variable **`tk::svgFmt`** (thanks to **Harald Oehlmann** for starting this action).

- Made sure that the values of a series of **`ttk::style`** options can be specified as arbitrary screen distances rather than in pixels only (thanks to **Jan Nijtmans** for making the majority of the necessary changes in the C code).

- Replaced most widget and **`ttk::style`** option values given in pixels in the Tk and Ttk library scripts with their counterparts in points (again, thanks to **Jan Nijtmans** for his valuable contribution). At the very few places where this was not possible, scaled the pixel values according to the value of the variable **`tk::scalingPct`**.

- Replaced the implementation of the indicators of the Tk core checkbutton and radiobutton widgets on X11 with a completely new one that uses scaling-aware SVG images (thanks to **Brian Griffin** for the very fruitful collaboration).

- Similar step for the indicators of the ttk::checkbutton and ttk::radiobutton widgets of the built-in themes **`alt`**, **`clam`**, and **`default`**, as well as for the ones of the Tk core checkbuttons and radiobuttons on Windows.

- Further changes in the C code: Made the cascade arrows of the menu entries on X11 and the indicators of the ttk::checkbutton and ttk::radiobutton widgets of the **`winnative`** theme scaling-aware.

- Made the mouse wheel bindings for the text widget (which trigger a scrolling by pixels) scaling-aware.

- Made the Widget Demo fully scaling-aware.

---

5. Developing Scaling-aware Applications with Tk 8.7

A few rules to follow in order to make sure that the sizes of the various GUI components of Tk 8.7 applications will automatically adapt themselves to the display's DPI scaling level:

- Specify font sizes in points only rather than in pixels.

- Whenever possible, use physical screen distances (in points, millimeters, centimeters, or inches) rather than their counterparts in pixels.

- If some calculations expect an amount in pixels, remember that you can transform points to pixels via

```
set pixels [expr {round($points * [tk scaling])}]
```

- Some computations might expect a value that is scaled according to the display's DPI scaling percentage. Here is how you can scale an integer:

```
set scaledNumber [expr {round($number * $::tk::scalingPct / 100.0)}]
```

- Within a canvas it might be more comfortable to work with cooordinates in pixels than to specify them as physical screen distances. In such cases you will want to rescale the coordinates of all of the items given by a tag or id with the aid of the **scale** canvas subcommand, according to the display's DPI scaling percentage. Here is how to compute the scale factor for both the x- and y-coordinates:

```
set scaleFactor [expr {$::tk::scalingPct / 100.0}]
```

- Whenever possible, use SVG images only and create them with the aid of the variable **tk::svgFmt**. If your application, for whatever reason, uses also raster images, then create them in several sizes, corresponding to the expected values of the variable **tk::scalingPct**. In this case it can come in handy to load the images from files having names of the form *myImg*100.png, *myImg*125.png, *myImg*150.png, etc.

---

6. How the Widget Demo Has Become Scaling-aware in Tk 8.7

The current trunk version of the **Widget Demonstration** application is completely scaling-aware, due to the following changes:

- Replaced the raster images used in the buttons **See Code**, **See Variables**, **Dismiss**, **Rerun Demo**, and **Print Code** with SVG icons from the Bootstrap project, created using the variable **tk::svgFmt** (thanks to **Alexander Schöpe** for the very useful link).

- Similarly, replaced the raster images used for the sort arrows in the script mclist.tcl with SVG icons, created using the variable **tk::svgFmt**.

- The red **NEW** indicator is no longer a raster image but a bold text in red color.

- Wherever it was possible, replaced the screen distances in pixels with their equivalents in points. The notable exceptions are as follows:

    - In the **Arrowhead Editor Demonstration**, implemented in the file arrow.tcl, there are a number of canvas coordinate computations, in which it was much easier to work with screen distances in pixels, scaled according to the value of the variable **tk::scalingPct**, than to replace everything with points.

    - The **Floorplan Canvas Demonstration**, implemented in the file floor.tcl, uses hundreds of canvas coordinates in pixels. Replacing them with their counterparts in points would have been extremely time-consuming, tedious, and error-prone. It was incomparably simpler to invoke the **scale** canvas subcommand, with a scale factor computed from the value of the variable **tk::scalingPct**. Same method applied to the much simpler demos **Knight's Tour** (knightstour.tcl), **Horizontal Scale Demonstration** (hscale.tcl), **Vertical Scale Demonstration** (vscale.tcl), and **Animated Wave Demonstration** (aniwave.tcl).

    - In the **Pendulum Animation Demonstration**, implemented in the file pendulum.tcl, replaced a series of screen distances originally given in pixels with their equivalents in points,

thus making them scaling-aware. Since various computations expect operands in pixels, at several places it was necessary to transform points to pixels, using the value `[tk scaling]`.

- The demo titled **Tk Goldberg (demonstration)**, implemented in the file `goldberg.tcl` (written by **Keith Vetter**), demonstrates not only "*how complex you can make your animations become*", but also how challenging it can be to transform a completely scaling-agnostic script, containing hundreds of hardcoded coordinates and other data of a lot of canvas items, into a scaling-aware application. While it was not hard to scale the item coordinates in *static* state with the aid of the **`scale`** canvas subcommand (just like in the script `floor.tcl`), for item options like **`-width`** or **`-arrowshape`** it was necessary to replace the values in pixels with their equivalents in points. Tiling the strike box of the match with the built-in bitmap **`gray25`** had to be postponed to the time after scaling the canvas item coordinates and has become more complex. The item coordinates in *dynamic* state, hardcoded in pixels, had to be replaced with their scaled counterparts, like in the script `arrow.tcl`. In addition, the window now fits on a display of full HD resolution and a scaling level of 150 %, due to optimizations regarding its height.

- For a better look of the scaled GUI, some of the widget demos now replace the photo images with magnified copies. Due to the way the zoom factor passed to the image **`copy`** subcommand is calculated, the copy will only be effectively magnified if the display's scaling percentage is at least `200`, which is always the case on a scaled display when running GNOME on Xorg or the Cinnamon desktop:

  ```
  set zoomFactor [expr {$tk::scalingPct / 100}]
  ```

  The affected demos are **Label Demonstration** (`label.tcl`), **Image Demonstration #1** (`image1.tcl`), **Text Demonstration - Embedded Windows and Other Features** (`twind.tcl`), **Canvas Item Demonstration** (`items.tcl`), **Printing Demonstration** (`print.tcl`), **Animated Label Demonstration** (`anilabel.tcl`), and **Window Icon Demonstration** (`windowicons.tcl`). Thanks to **Paul Obermeier** for his tip related to zooming a multi-part GIF image, used in the script `anilabel.tcl`.
