TileQt and TileGtk: current status

Georgios Petasis

Software and Knowledge Engineering Laboratory,
Institute of Informatics and Telecommunications,
National Centre for Scientific Research "Demokritos",
Athens, Greece
petasis@iit.demokritos.gr

Abstract

This paper is about two Tile and Ttk themes, TileQt and TileGTK. Despite being two distinct and very different extensions, the motivation for their development was common: making Tk applications look as native as possible under the Linux operating system.

1 Introduction

In this paper we describe two extensions, TileQt and TileGTK, which aim to provide native look to Tk applications that use Tile/Ttk under the Linux operating system. Ttk is the best effort so far in providing Tk widgets with native look under all major operating systems. Among these, Microsoft Windows and Apple OS X offer a native widget set, and a suitable API, that applications can use in order to natively draw widgets. Wisely, Ttk fully exploits these APIs, allowing Tk applications to look exactly as native applications under these two operating systems.

The situation is very different for the Linux operating system, though. Under Linux, there is no native API that can be used by applications in order to draw widgets. Instead, the Linux desktop is mainly dominated by two environments, KDE [6] and Gnome [5], which are based on two different toolkit libraries, Qt [3] and GTK [4] respectively. These two libraries follow a quite different approach in implementing widgets, making the two libraries incompatible with each other. Both Qt and GTK support styled widgets, following again a different approach on customising the visual appearance of widgets. What is common between the two however, is the difficulty for third-party software to access their style API: both assume that applications use one of the libraries as their widget implementation basis, frequently mixing widget styling details with their internal implementation details.

Both TileQt and TileGTK try to use (and in some cases abuse) what is available as public APIs in the respective widget toolkits. The public APIs usually target either style development (so as new widget styles or themes can be developed) or new widget development (so as new or composite widgets can be developed). Despite the different scope of the public APIs, both TileQt and TileGTK try to use the available public APIs in order to draw widgets, a job that is usually performed by the toolkit internal (and thus not exported) functionality. This is why the development of both extensions followed a similar path: the sources of Qt and GTK were studied, in the detail required to understand how each library draws its widgets. Then, each extension tries to reproduce, as much as possible, the drawing behaviour of the corresponding toolkit, but organised in the framework defined by Ttk. Of course, it is not always easy, or even possible, to reproduce the internal drawing operations of another toolkit, mainly because the public API is not sufficient for this purpose. Typical examples are the scale/slider widgets from the Qt toolkit. These widgets seem as an ad hoc addition to the library: instead of following an approach that decomposes these widgets into elements, the widgets are drawn as a whole, by a single function in the context of an object of a specific

C++ class. An approach like the aforementioned one is difficult to be reproduced without simulating a large part of the toolkit. Thus, in such cases, both TileQt and TileGTK follow a different approach:

- Instances of widgets are created by using the public API of the widget toolkit.
- The properties of these widget instances are modified to match the Ttk element that must be drawn.
- The widget instances are drawn on an offline (not visible) pixmap.
- The pixel metrics of the widget instances are used in order to locate the widget part/element that corresponds to the desired Ttk one.
- The located part/element is copied onto the window that must be drawn by the Ttk element.

Of course, deviations from the expected behaviour are expected, when a public API is used for purposes other than the ones it was designed for. It is not uncommon to receive a wrong answer for a query on a theme property not frequently requested by the widget toolkit when drawing a widget, or to need a property that cannot be retrieved from the toolkit theme. In such a case, usually there is no alternative than to improvise. And when properties are guessed, their value can never be correct for all cases and for all themes, leading to visual inconsistencies and unsatisfied users.

The rest of this paper is organised as follows: in chapter two the problems of interfacing a toolkit library to Ttk are presented, followed by a brief presentation of the TileQt and TileGTK extensions, along with the supported Ttk widgets by each extension. Finally, some open issues and future directions are presented in chapter three, which concludes this document.

2 Mapping between Ttk and other widget toolkits

TileQt and TileGTK are quite different from each other, implementation wise. However, the problems of interfacing Ttk to another toolkit library (that being Qt, GTK or any other library) are exactly the same, and seem to be independent of the target toolkit library:

- Understand the internals of the library that must be interfaced. This is a time consuming, but feasible task, if the library sources are available. Thankfully, the sources of both Qt and GTK are publically available, along with sufficient documentation.
- Understand how to initialise the library from a hosting application. Both the Qt and GTK libraries assume that they are in control of the application, and that their “main” function will be executed, which will usually initialise the library and install a never-ending message loop. Of course, calling a function that never returns is not a desirable action, thus it is important to find a way to initialise the library without calling functions that will block (i.e. because the event loop is started and its termination usually means the termination of the application).
- Understand how the toolkit library locates themes, and how themes are loaded and used in order to draw widget elements.
- Find a way to map Tk drawables (windows, pixmaps, etc.) to the drawables of the toolkit library, and vice versa. This is an important step, as each library expects its own structures while using its API. This task may be difficult to be achieved, if the implementation is limited only to the public API of the library (targeting portability), as all libraries tend to hide platform-specific structures under abstraction layers, in order to ensure portability across multiple operating systems.

- Find a way to map Ttk widget states to the ones supported by the target toolkit library. This mapping is not always straight-forward, and mapping differences usually result in visual differences.
- Identify which widgets can be drawn directly (i.e. widget elements can be mapped between Ttk and the target toolkit library, through its theme API), and which widgets have to be drawn as a whole, followed by a segmentation step so as to identify and extract individual elements. Widgets belonging to the latter category are always difficult to be supported correctly, as it is not uncommon the pixel metrics of elements to be erroneously declared by the theme, especially since they are rarely needed (as the widget is drawn as a whole by the toolkit). This is quite common in Qt: being written in C++ it is easy for a theme to just subclass another theme, draw the widget differently, but never update the methods that return size information for the various elements.
- Ensure thread-safety. This can be an easy task if the target toolkit library is already thread safe (i.e. Qt where widgets can be created and drawn by any thread), or more tricky if it is not (i.e. GTK, where widgets must be created/drawn only by the thread the library was initialised).

And of course, an extension that is a Ttk theme must cope with the large number of available themes for the interfaced library. It is common that some themes work as expected, while others don't, due to various reasons: from using a new widget layout (i.e. scrollbars) to reporting erroneous pixel metrics for elements.

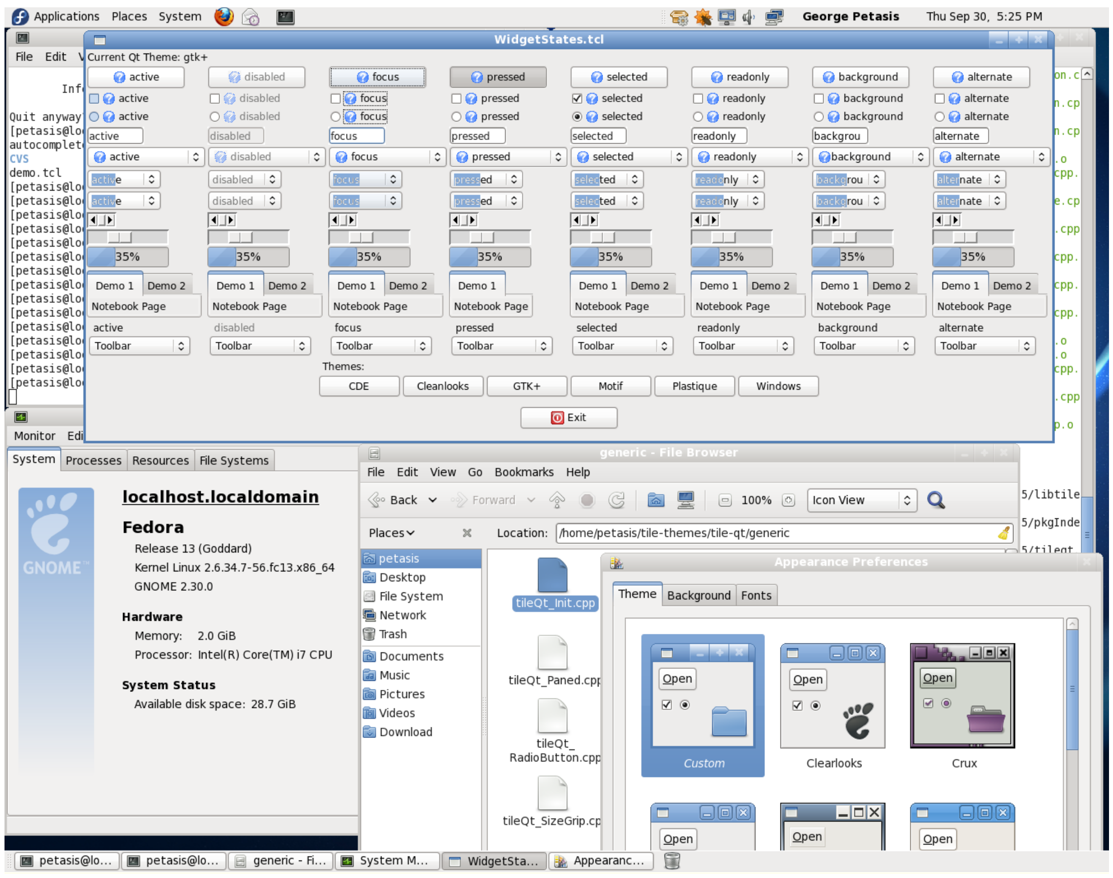
2.1 TileQt: a Ttk interface to the Qt widget toolkit

TileQt is a Tile/Ttk theme that aims to use Qt styles for drawing Ttk widget elements. Historically, it was the first C/C++ extension that attempted to provide a Ttk theme, based on the Ttk public API, without being part of Ttk. Its development started around 2003, supporting only Qt 3.x at that time. As a first attempt, several obstacles had to be overcome, ranging from initialising Qt and loading the style along with Tk, to building the extension with TEA/autoconf, which has limited support for locating the C++ compiler or Qt libraries. Several attempts were made by the author to build a robust build system for TileQt, from plain autoconf scripts, to a combination of Tcl scripts that locate components and then drive configure. The fact that all of them failed was the main motivation for switching the build system to CMake [1], which currently is able to support all the build requirements of TileQt. Also, TileQt was extended to support Qt 4.x in addition to Qt 3.x, when KDE 4.x was released: TileQt aims to provide support for both Qt 3.x and Qt 4.x, as well as Tile in Tk 8.5 and Ttk in Tk 8.5/Tk 8.6. In general, supporting both Tile and Ttk is easy, with the exception that TileQt cannot be built with the latest ActiveTcl 8.6 beta 2 distribution, due to an outdated version of "ttkDecls.h". Instead, a newer version must be retrieved and used, from the Tk CVS server, hosted at SourceForge [2].

TileQt has not been maintained for quite some time now (the last ChangeLog entry dates back to December 2008), but the source code compiles under an up-to-date Linux distribution (tested with Fedora 13 and the latest 8.6 beta from ActiveState [7]). Invoking one of the TileQt demos produces the output shown in Figure 2, while the Ttk elements and widgets supported by TileQt are shown in Figure 1. As it is evident from both figures, TileQt currently lacks support for scrollbar and scale widgets under Qt 4.x, as the implementation has not been updated from the Qt 3.x era. Looking closely at Figure 2, more display inconsistencies can be detected, such as faint white traces at the round corners of entry widgets, and offsets by one or two pixels on notebook tab rendering. However, these inconsistencies are not so important or noticeable by end users, at least compared to the complete lack of support for scrollbars, an essential widget for many applications. Finally, a problem seems to exist regarding thread safety of TileQt with recent Tk versions: the basic demo script (demo.tcl) of TileQt crashes when a second interpreter is initialised, so as to display a Qt theme selection dialog, which suggests important changes in recent Tk/Ttk versions that TileQt must take into consideration.

Widget	Qt 3.x	Qt 4.x	Widget	Qt 3.x	Qt 4.x
Background	☑	☑	LabelFrame	☑	☑
Button	☑	☑	NoteBook	☑	☑
CheckButton	☑	☑	TreeView	☑	☑
RadioButton	☑	☑	Progress	☑	☑
MenuButton	☑	☑	Paned	☑	☑
ToolButton	☑	☑	SizeGrip	☑	☑
Entry	☑	☑	ScrollBar	☑	☒
ComboBox	☑	☑	Scale	☑	☒

Figure 1: Widgets supported by TileQt under Qt 3.x and Qt 4.x.



localhost.localdomain

Fedora

Release 13 (Goddard)
Kernel Linux 2.6.34.7-56.fc13.x86_64
GNOME 2.30.0

Hardware
Memory: 2.0 GiB
Processor: Intel(R) Core(TM) i7 CPU

System Status
Available disk space: 28.7 GiB

Location: /home/petasis/tile-themes/tile-qt/generic

Figure 2: TileQt running under the Gnome Desktop, in an up-to-date Fedora 13 Linux distribution.

2.2 TileGTK: a Ttk interface to the GTK widget toolkit

TileGTK is a Tile/Ttk theme that aims to use GTK styles for drawing Ttk widget elements. TileGTK is a recent extension, compared to TileQt, as its development started on August 2008. However, since both extensions were developed by the same author, TileGTK shares the same approach as TileQt: its development started from the TileQt sources, adapting names with search and replace, and slowly porting the widgets from Qt to GTK, one by one. The CMake-based build system was also inherited from TileQt. Of course, the development of TileGTK involved an extensive examination of GTK and GLib sources, in order to understand how GTK draws its styled widgets, revealing a significantly different approach than Qt. A major cause for this difference is the fact that GTK is written in C, instead of the

C++ employed by Qt. Of course, interfacing to a C library has some advantages, like the ability to use the library through a stubs interface, which TileGTK employs: reusing the GLib facilities for loading dynamic libraries, TileGTK is linked only to GLib. GTK is located and loaded dynamically, when the TileGTK extension is initialised from Tcl. And assuming that Tcl will add dynamic loading of arbitrary shared libraries in the (hopefully near) future, TileGTK can drop linking to GLib, allowing binary distributions of TileGTK to be distributed under the BSD license, to match the license of the sources.

Again TileGTK has not been actively maintained for some time now (the last ChangeLog entry dates back to August 2009), mainly because the usage of the Linux operating system by the author has been significantly diminished, compared to previous years. However, the source code still compiles under an up-to-date Linux distribution (tested with Fedora 13 and the latest 8.6 beta2 from ActiveState). Invoking some of the TileGTK demos produces the output shown in Figure 4, while the Ttk elements and widgets supported by TileGTK are shown in Figure 3. As it is evident from both figures, TileGTk currently lacks support for the treeview widget, as the implementation has never been written. Looking closely at Figure 4 (and Figure 3 for gray-coloured checked items), reveals several visual inconsistencies for some widgets. For example, the arrow for comboboxes and menubuttons cannot be drawn through the available public API, thus a generic arrow is drawn instead, borrowed from the scrollbar widget. Despite the fact that the scrollbar widget can be split in elements, and elements can be drawn by using the public API of GTK, there are visual differences in the arrow background shapes (they are not round, like the ones shown in the native GTK scrollbar visible at the right top corner of Figure 4). Scale and progress widgets are also drawn wrongly: scale grips are drawn larger than the native ones, suggesting a segmentation error while decomposing a widget drawn as a whole by the GTK style engine into elements. Similarly, the filled area of progress widgets overlaps with the widget borders. Finally, there exists a white line, drawn just below the raised tab in notebook widgets. In addition, the problem of thread safety observed in TileQt is also present in TileGTK: the basic demo script (demo.tcl) of TileGTK crashes when a second interpreter is initialised, so as to display a GTK theme selection dialog.

Widget	GTK 2.x	Widget	GTK 2.x
Background	✅	LabelFrame	✅
Button	✅	NoteBook	☑
CheckButton	✅	TreeView	❌
RadioButton	✅	Progress	☑
MenuButton	☑	Paned	✅
ToolButton	✅	SizeGrip	✅
Entry	✅	ScrollBar	☑
ComboBox	☑	Scale	☑

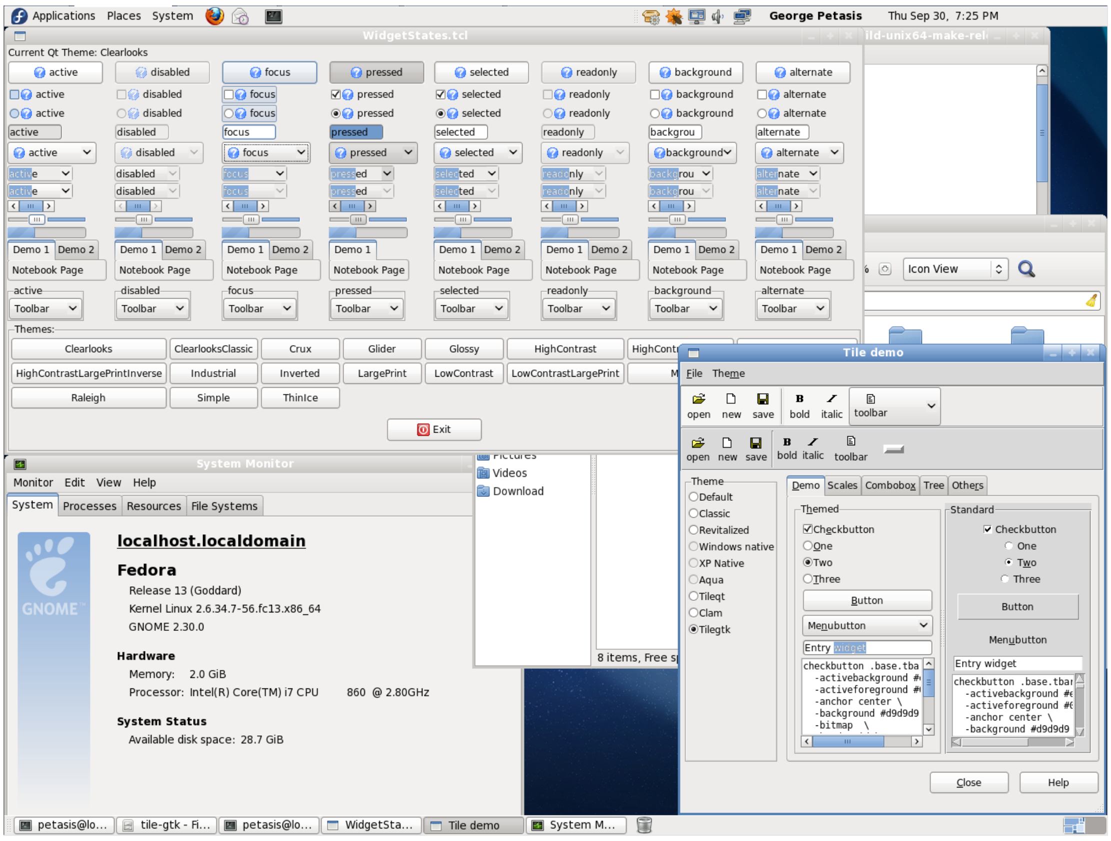
Figure 3: Widgets supported by TileGTK under GTK 2.x.

3 Conclusions, Open issues and Future directions

Maintaining an extension such as TileQt and TileGTK is not an easy task. Widget toolkits tend to evolve over time, sometimes introducing incompatible changes (i.e. Qt 4.x vs Qt 3.x), or deprecating useful functions. Apart from the constant monitoring required, there is also the issue of themes: there are too many of them floating around. It is not uncommon to fix a drawing error in one theme, with a change that breaks another theme, leading to a dead end. A promising idea is, of course, to prioritise themes: default themes of popular Linux distributions (such as Ubuntu or Fedora) should have priority over the rest of available themes.

Another aspect is the distribution license of generated binaries. The source code is distributed under the BSD license, but BSD does not cover the produced binaries. Both TileQt and TileGTK need to link with toolkit libraries (Qt, GLib, GTK) that are covered by the GPL license, which is transferred to the produced binaries. The fact that GTK is written in plain C, allowed TileGTK to access almost all GLib and GTK functions through a stub table. The only functions used directly are the ones related to shared library loading, offered by GLib. Once these functions are replaced by equivalents offered by Tcl, then linking to GTK will only happen at runtime, offering the possibility for TileGTK binaries to be delivered under a different license, such as BSD.

Finally, integration with the dominant Linux desktops, Gnome and KDE, is still an issue. Retrieving the theme, currently used by the desktop, is not always easy (especially for Gnome), while the retrieval of the colour scheme can be even more problematic. Regarding the latter, Qt and KDE provide more facilities towards this goal, while GTK doesn't. Under GTK, the missing functionality is attributed by parsing GNOME configuration files, and despite not being an elegant solution, it seems to work for the time being. Future work on both TileQt and TileGTK should focus on implementing the missing widgets (i.e. scrollbars under Qt 4.x) as well as fixing the visual problems that can be observed under popular themes.



```
checkbutton .base.tba
  -activebackground #
  -activeforeground #
  -anchor center \
  -background #d9d9d9
  -bitmap \
```

localhost.localdomain

Fedora

Release 13 (Goddard)
Kernel Linux 2.6.34.7-56.fc13.x86_64
GNOME 2.30.0

Hardware

Memory: 2.0 GiB
Processor: Intel(R) Core(TM) i7 CPU 860 @ 2.80GHz

System Status

Available disk space: 28.7 GiB

Figure 4: TileGTK running under the Gnome Desktop, in an up-to-date Fedora 13 Linux distribution.

4 References

[1] CMake – A cross-platform make: http://www.cmake.org/
[2] Tk Toolkit CVS access: https://sourceforge.net/scm/?type=cvs&group_id=12997
[3] Qt – A cross-platform application and UI framework: http://qt.nokia.com/
[4] The GTK+ project: http://www.gtk.org/
[5] GNOME: The Free Software Desktop Project: http://www.gnome.org/
[6] The KDE Community: http://www.kde.org/
[7] ActiveState – The Dynamic Language Experts: http://www.activestate.com/