LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Information Technology
Master's Thesis

GTK+ toolkit on mobile Linux devices

The topic of this Master's thesis has been approved by the board of Department of Information Technology on November 9, 2005.

Supervisor: M.Sc. Pasi Nieminen
Examiner: Prof. Jari Porras

Lappeenranta, December 18, 2005

Kaj Grönholm
Nelikkotie 8 A 4
FIN-02230 Espoo
+358 (050) 53 08 233
kaj.gronholm@nomovok.com

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology
Kaj Grönholm

GTK+ toolkit on mobile Linux devices

Master's Thesis
2005

89 pages, 32 figures, 8 tables and 2 appendices

Examiner: Professor Jari Porras
Keywords: GTK+, Linux, embedded, mobile devices, user interface, toolkit, optimization, cross-compile, CPU-transparency

Linux operating system (OS) is widely used in the server and desktop computers. Recently the popularity of Linux has increased also in the embedded devices, such as PDAs, mobile phones and different industry equipments. Due to screen size, performance and usability, these devices have special needs for their graphical user interface (GUI). There are multiple GUI toolkits available for Linux, GTK+ being one of the most popular one.

This Master's thesis gives an introduction to embedded Linux and to GTK+ toolkit, evaluating how well they suit on mobile devices. One of the biggest obstacles in bringing desktop technologies such as GTK+ to mobile devices is the performance. A tool to test and profile this performance called GtkPerf was implemented as a part of this thesis.

The conclusion of this work is that with a reasonable amount of modifications and optimizations, GTK+ can successfully be used in embedded devices. The first commercial mobile device based on GTK+ toolkit, Nokia 770 Internet Tablet, is an evidence of this.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto
Kaj Grönholm

GTK+ toolkit on mobile Linux devices

Diplomityö
2005

89 sivua, 32 kuvaa, 8 taulukkoa ja 2 liitettä

Tarkastaja: Professori Jari Porras
Hakusanat: GTK+, Linux, sulautettu, mobiilit päätelaitteet, käyttöliittymä, käyttöliittymäkirjasto, optimointi, ristiinkääntäminen, suoritin-läpinäkyvyys

Linux -käyttöjärjestelmä on laajasti käytössä palvelin- ja työpöytätietokoneissa. Linux on lisäämässä suosiotaan kuitenkin myös sulautetuissa laitteissa, kuten PDA:issa, kännyköissä sekä erilaisissa teollisuusjärjestelmissä. Näytön koko, suorituskyky ja käytettävyys asettavat omia erityisiä tarpeitaan laitteiden graafiselle käyttöliittymälle. Linux -käyttöjärjestelmälle on olemassa useita käyttöliittymäkirjastoja, joista GTK+ on yksi yleisimmin käytetyistä.

Tämä diplomityö esittelee sulautetun Linux -käyttöjärjestelmän ja GTK+ käyttöliittymäkirjaston, selvittäen miten hyvin ne soveltuvat mobiileihin päätelaitteisiin. Yksi suurimmista esteistä työpöytäkäyttöön suunnattujen teknologioiden, kuten GTK+, muokkaamisessa mobiileihin päätelaitteisiin on suorituskyky. Osana tätä työtä kehitettiin GtkPerf -niminen työkalu, jolla GTK+:n suorituskykyä eri alustoilla pystytään helposti mittaamaan ja havaitsemaan mahdollisia pullonkauloja.

Tämän työn johtopäätöksenä on, että pienillä muokkailuilla ja optimoinneilla GTK+ soveltuu myös mobiileihin päätelaitteisiin. Ensimmäinen kaupallisesti saatavilla oleva GTK+ -pohjainen päätelaite, Nokia 770 Internet Tablet, on tästä osoituksena.

PREFACE

This thesis was written for Nomovok Ltd. at OS Embedded Solutions department. Writing had started already at the beginning of spring 2004, but due to workload (on the same subject) the thesis was left on lower priority. When the release of Nokia 770 Internet Tablet finally came on May 2005, there also came time to write down all the lessons learned during this intensive time period and this thesis got finalized.

I would like to thank the following people for their help in this process: Professor Jari porras, for your guidance and for pushing/kicking me to finalize this work. Pasi Nieminen and Aki Kolehmainen from Nomovok, for your professional help and open mind towards this thesis. My parents, for believing in me and for giving your overwhelming support during my studies in Lappeenranta.

Special thanks to my beloved wife Maija, for your never ending love and understanding. Together with Eelis, You rock my world!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ABI	Application Binary Interface
API	Application Programming Interface
ARM	Advanced RISC Machines
ATK	Accessibility Toolkit
BSD	Berkeley Software Distribution
CPU	Central Processing Unit, processor
GDK	GTK+ Drawing Kit
GNOME	GNU Network Object Model Environment
GNU	GNU's Not Unix
GPL	General Public License
GTK+	The GIMP Toolkit
GUI	Graphical User Interface
LGPL	Library General Public License
MMU	Memory Management Unit
MVC	Model – View – Controller
NFS	Network File System
NPTL	Native POSIX Thread Library
OS	Operating System
PDA	Personal Digital Assistant
POSIX	Portable Operating System Interface
RAM	Random-Access Memory
RISC	Reduced Instruction Set Computer
SDK	Software Development Kit
SVG	Scalable Vector Graphics
TCP/IP	Transmission Control Protocol / Internet Protocol
UI	User Interface
UNIX	Operating System designed in AT&T Bell Laboratories
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
X11	X Window System version 11

1 INTRODUCTION

Linux operating system was first released in 1991 and since then it has seen wide embrace among users. It has grown from a small Unix-like clone to mature and solid operating system for servers, workstations and clusters.

Linux has a good possibility to revolutionize the whole operating systems genre. Historically there have been different environments for different systems, servers, desktops, mobile devices etc. but now they can potentially be all replaced with one free kernel. Linux operating system is today more popular than ever before and as it gets larger user base it also gets adapted to new platforms. One of them is in embedded devices.

There are many types of embedded systems, from which this thesis concentrates on small mobile devices such as PDAs and higher-end mobile phones also known as smartphones. One requirement of these devices it that they need a graphical user interface. This master thesis evaluates one popular graphical toolkit for Linux, Gimp Toolkit (GTK+).

There exists a lot of information about GTK+ around the Internet, but most of the knowledge is not as current and accurate as one would wish. Best sources of GTK+ related information are the API documentation in GNOME website /1/ and the only recent book written about the subject, GNOME 2 Developer's Guide /2/. All other GNOME and GTK+ books are based on older GTK+ 1.x which differs greatly enough from the current 2.x series to make them quite irrelevant.

This thesis does not try to replace those information feeds, but to complement them and add another twist: How well does Linux and GTK+ suit the needs of embedded devices operating system and user interface?

Using GTK+ in embedded world has been very minimal until the recent introduction of maemo, Linux development platform created for Nokia 770 Internet Tablet /3/. Maemo plays also a big part of the reason this thesis exists.

1.1 Structure of the work

This thesis starts with an introduction to Linux operating system and proceeds from there to GTK+ toolkit. The mobile aspect is carried trough out the thesis, but last chapters are emphasizing it even more. Structure of the chapters is following:

Chapter 1 - This introduction.

Chapter 2 - Introduces Linux operating system and how it suits to embedded device usage. Includes comparison to other mobile operating systems.

Chapter 3 - Goes through the challenges of developing applications to small devices. Cross-compilation using Scratchbox is explained.

Chapter 4 - Introduction to GTK+ graphical toolkit. Its design and all related libraries are gone through with the intention of explaining how they relate to each other and which library is in charge of what. Contains also information about customizing GTK+ UI with themes.

Chapter 5 - Introduces GTK+ GUI widgets. Being one of the main aspects of this thesis, this chapter is the longest one even thought it doesn't contain all the seldom used widgets. Lots of code examples and screenshots are included for clarification.

Chapter 6 - Suitability of GTK+ in mobile devices. Performance testing to clarify how the speed of GTK+ suits to limited devices and which parts need most of the optimization. Performance measurement tool, GtkPerf, which was developed as part of this thesis is introduced with the results.

Chapter 7 - Results and the conclusions of the work.

2 LINUX AS A MOBILE OPERATING SYSTEM

This chapter compares the most popular mobile operating systems and explains how Linux fits in that group. Different embedded Linux distributions, both open and commercial, are listed in the end of the chapter.

2.1 Requirements of a mobile operating system

All computer devices, whether they are big mainframe servers or small special purpose devices, need some kind of operating system. It is obvious that these different platforms require different support from their system software. Something which is important in server operating system, can be totally unimportant and wrong for embedded OS.

Some basic requirements especially for operating systems used in mobile devices such as PDAs are:

- Low hardware requirements for CPU, amount of memory and electric power consumption.
- Easily configurable to work with different hardware setups like processors, memory and screen sizes.
- Low-level device drivers and support available from hardware manufacturers. If this support is not available, open source community needs at least decent hardware documentation to be able to maintain these drivers.
- Modular structure, so that operating system can be extended to different directions. This was not as important in older systems but nowadays devices are so powerful that modularity becomes essential.
- Power saving options to increase the battery life of the device. This extends the mobility of the device and so that it can be used for longer periods without recharging.
- Support for different connectivity technologies like TCP/IP and USB. Usually also wireless ones such as WLAN, Irda and Bluetooth should be supported.

On top of these requirements, embedded operating systems need the same aspects of a good OS, like powerful platform libraries to build applications. After all, applications are the thing which pull though the OS; no matter how perfect the operating system is, without good applications it doesn't survive.

One very important issue aside the technical features where Linux especially shines, is the price. Because of the licensing fees, most of the embedded operating systems can become quite expensive for the device manufacturers. With Linux this is not the case as the main parts of the system are free in both meanings of the word: It doesn't cost anything to get and use Linux, and it can also be freely modified to suit the specific needs.

2.2 Comparison of available mobile operating systems

There exists many operating systems designed to be used in the mobile devices targeted in this thesis. Even thought Linux has a good chances to be the dominant platform in the future, many other OS's are currently fighting for their shares. Figure 1 shows the market shares of smartphone operating systems during the first quarter of 2005 based on research by Gartner /4/.

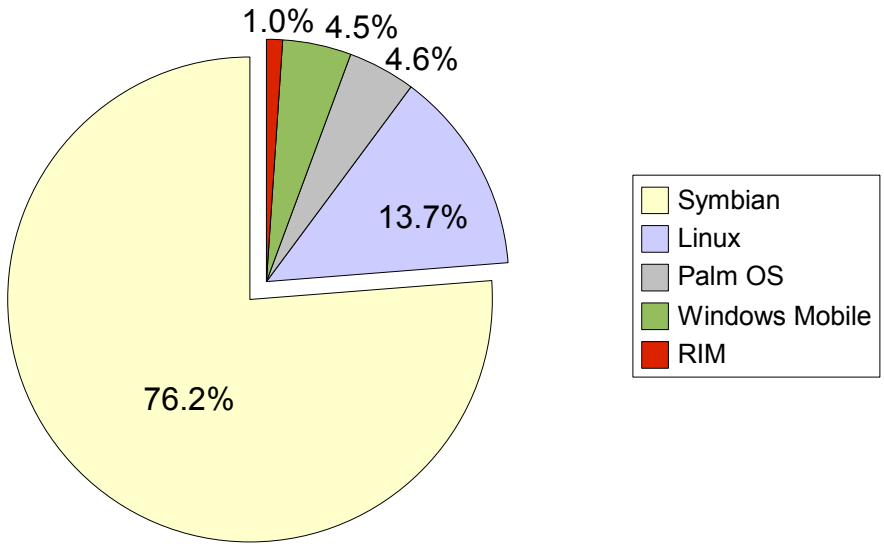


Figure 1. Smartphone shipments in Q1 2005. Source: Gartner

Symbian leads the Smartphone shipments with a 76.2% market share, followed by Linux with 13.7%. Linux is popular especially in China and other eastern countries, which are currently the fastest growing market areas.

Another research from IDC reports that in converged mobile device operating systems Symbian leads with 55.9%, followed by Microsoft Windows Mobile with 12.7% and Linux with 11.3%. Same research estimates that Linux will have nearly 17% market share by the year 2009. The difference between these studies can be explained with the fact that former includes only smartphones and not Microsoft Pocket PC PDAs /5/.

Next sections will introduce these most prominent competitors of Linux OS.

2.2.1 Symbian

Symbian is a software licensing company that develops and supplies the Symbian OS, designed for data-enabled mobile phones. Symbian was established as a private independent company in June 1998 by Ericsson, Nokia, Motorola and Psion. Nowadays the biggest owners are Nokia (47.9%), Ericsson (15.6%), Sony-Ericsson (13.1), Panasonic (10.5%), Siemens (8.4%) and Samsung (4.5%) /6/.

Symbian has a very good market share right now mainly thanks to Nokia, its biggest owner and supporter. But the future does not necessarily look so bright as the share now suggests.

- Symbian is designed for quite small niche of embedded devices, multimedia phones. It is not used in others like PDAs and media terminals.
- Symbian is based on old *Psion Epoc* operating system, and its native programming language is a non-standard C++. This makes the learning curve and developing for Symbian quite hard compared to other platforms.
- Symbian as a platform does not scale too well for newer devices with bigger screen sizes, memory's and more processing power. As devices become more powerful, easier development and modularity becomes more important than getting every piece of power out of the system.

- Nokia's share of Symbian has been growing lately as Motorola and Psion sold their shares /6/. Other manufacturers are not so happy about this situation where Nokia has too big domination on direction of Symbian.

Symbian is leading the multimedia phones OS war now, but it may very well be possible that it doesn't scale to future needs as well as would be required and manufacturers start to seek for other solutions.

2.2.2 Windows Mobile

Windows Mobile is Microsoft's software platform for Pocket PC PDAs and smart phones. Windows Mobile extends the familiarity of the Windows desktop to personal devices. There are now three common Windows Mobile software releases, Windows Mobile 2002, 2003 and 2003 Second Edition /7/.

Microsoft has tried for a long time to get a footprint on mobile OS market, and finally their investments are paying off. As Figure 1 showed, Microsoft has the third best selling smartphone operating system currently. Strongest points of the platform are also partly its weaknesses:

- Microsoft is a very strong company, with revenue of over $32 billion in 2003 /7/. This makes them an noteworthy competitor in any line of business they choose.
- Thanks to Microsoft's clear leadership (some might even call it a monopoly) in desktop PC operating systems, Windows Mobile has very good connectivity and synchronization features.
- Microsoft is not used to build operating systems to devices with limited resources. This has been the biggest obstacle why Windows Mobile has not managed to become more popular earlier. Now when these devices are more and more powerful, this has become a smaller issue.
- Hardware companies are afraid that Microsoft will get the same kind of monopoly in mobile devices as they have in desktop computers, and are therefore not too eager to jump using their OS.

Technically the Windows Mobile is becoming ready, and the future of it is mostly up to the device manufacturers.

2.2.3 Palm OS

Palm OS is most widely used PDA operating system, even thought Microsoft has already taken a bigger part in current sales. Palm Computing was founded in 1992 and acquired by U.S. Robotics Corp. in 1995. In 1996, Palm introduced the Pilot 1000 and Pilot 5000 products that led the resurgence of handheld computing. In June 1997, Palm became a subsidiary of 3Com Corp. when U.S. Robotics was acquired by 3Com. Later in September 1999, 3Com announced plans to make the Palm subsidiary an independent, publicly traded company and Palm OS was separated to its own company, PalmSource /8/.

Reason for Palm OS success has been it's pioneering in whole PDA market. Palm OS has never thought to have superior lower level platform, but the usability of the UI and wide selection of applications leaded it to own majority of PDA markets in the end of 90's. Since then the competition has grown and Palm has faced problems scaling their OS to more powerful hardware.

PalmSource acquired recently company called China MobileSoft (CMS). At the same time is announced that it's going to start using Linux as the basement for Palm OS. This means basically that Palm starts using Linux kernel and lower level libraries, and provide the user interface and wide application selection on top of that /9/.

With this change, Linux gets support from one of the biggest PDA companies. It will most likely help to convince hardware manufacturers to write drivers for Linux, but it's too early to know what kind of future Palm OS will have as a proprietary environment on top of that.

2.3 Linux as an embedded platform

2.3.1 Introduction to Linux

Linux is a clone of the UNIX operating system, written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Internet. It aims towards to full POSIX and Single UNIX Specification compliance. Linux has all the features one would expect from a modern fully-fledged Unix, including true multitasking, virtual memory, shared libraries, on-demand loading, shared copy-on-write executables, proper memory management, and multistack networking including IPv4 and IPv6 /10/.

Linux is not originally designed as an embedded operating system, but it has a very customizable kernel. This, combined with large amount of tools freely available, makes it very powerful platform to build on.

2.3.2 Linux kernel

Kernel is the heart of every operating system, also in Linux. There are several things to consider when selecting and configuring kernel for mobile Linux system. This thesis does not try to explain Linux kernel in details, only the most important achievements lately in the kernel development for embedded usage.

Linux kernel version 2.6 which was released at the end of 2003 was a major leap in many aspects. It improved support to both directions; in significantly larger systems and also to smaller ones. Best additions in 2.6 to suit mobile development are /11/:

- More modular approach of the whole kernel. It is now made up of components that are well separated, leading to easier implementation and customizing possibilities.
- With Linux 2.6, the kernel is now preemptible to a degree, making 2.6 more responsive than 2.4 and giving implementors better control over the timing of events. Kernel code has been salted with preemption points that enable the

scheduler to run and possibly block a current process so as to schedule a higher priority process.

- Kernel 2.6 features a rewritten process scheduler developed to eliminate the slow algorithms of past versions. Previously, the scheduler had to look at each ready task and score its relative importance. The time required for this algorithm varied with the number of tasks and thus complex multitasking applications would suffer from slow scheduling. In Linux 2.6, the scheduler no longer scans all tasks each time. Instead, when a task becomes ready to run it is sorted into position on a queue called the current queue.
- Improved synchronization support with fast user-space mutexes, which check from user space to see if blocking is necessary and only perform the system call when blocking the thread is required. These functions also use scheduling priority to decide which thread gets to execute when there is contention.
- Better support for newer technologies like Bluetooth, WLAN and USB 2.0.
- Includes Native POSIX Thread Library (NPTL) which is a significant improvement over the older LinuxThreads approach. Along with POSIX threads, 2.6 provides POSIX signals and POSIX high-resolution timers as part of the mainstream kernel.
- The Linux 2.6 version supports several current microcontrollers that don't have memory management units (MMU). These include Motorola m68k processors such as Dragonball and ColdFire, as well as Hitachi H8/300 and NEC v850 microprocessors. This support became with merging much of the uClinux project into the mainstream kernel.

Since the initial release of kernel 2.6, development has been very active on improving it and new main version branch has not been started yet.

2.3.3 What makes Linux competitive

There are various reasons to select Linux as an embedded platform over some other more traditional embedded OS.

Scalability

Same Linux kernel and architecture scales with small changes to both embedded usage and to big mainframe servers. Also the same development tools can be used in both of them. This can substantially decrease training and development costs and eliminate the need to learn a whole new OS if a product grows (or shrinks) more than expected.

Quality of code

Most programmers agree that Linux kernel and most projects used in a Linux system have good quality. The main reason for this is the open source development model, which gives all developers and users possibility to check out the source codes.

Open source community has a coding style where each separate functionality should be found in a separate module. This modularity helps to locate bugs and fix them in a single location. Linux system is also easily configurable to include only the system components needed.

Availability of code

Since Linux kernel and most of the normal Linux system applications are open source, the code is available for everyone. This makes chasing bugs much easier than with traditional closed source embedded OSes.

One thing that helps also the code quality is the GNU General Public License (GPL). This license, which is the most widely used license in Linux applications, has a feature which forces developers to distribute full source code with applications. This means that everyone has access to the source code and can make changes in there, but if they redistribute the application, full source code with all the changes needs to be made available too /12/.

GPL license can be problematic for enterprises who may want to keep their software closed, but must make their source codes available because of its viral effect. These are also other open source licenses (LGPL, BSD, MIT, etc.) with different obligations.

Hardware support

Linux has broad hardware support for different hardware platforms and devices. Although a number of vendors don't yet provide Linux drivers, considerable progress has been made recently. Large number of drivers are maintained by Linux community itself, which is so extensive that no other OS provides this level of portability. Thanks to open source, driver which has been written to one platform can usually quite easily be ported to another architecture Linux runs on.

Community support

Support from community is perhaps one of the biggest strengths of Linux. Every time you have some problem, it is likely that someone has had same kind of problems. Often this person will gladly provide the solution for you, or has already documented procedures getting past these difficulties to somewhere in Internet. Best places to get community support are multiple mailing lists and forums.

Vendor independence

Vendor independence means that users don't have to rely on any sole vendor to get or use Linux. Furthermore, if one is displeased with the vendor, it is possible to switch because the licenses under which Linux is distributed provide others the same rights as the vendors. Although some vendors provide additional software in their distribution which isn't open source and can become problematic when switching. For this reasons closed source parts should usually be avoided whenever possible.

Costs

The last reason to consider Linux adoption is the costs. There are mainly three components of costs in building a traditional embedded system: initial development setup, additional tools, and runtime royalties. With Linux this cost model is very different. All OS components and development tools are available free of charge, their license even prevents the collection of any royalties on these core components. There are of course also non-free components available, but they are not required.

Figure 2 shows the results of recent popular embedded Linux site survey “Which factors has had greatest influence for selecting Linux (as the embedded OS of your

choice)". From this graph it becomes obvious that freeness, availability of tools and support from community are the biggest reasons to go with Linux /13/

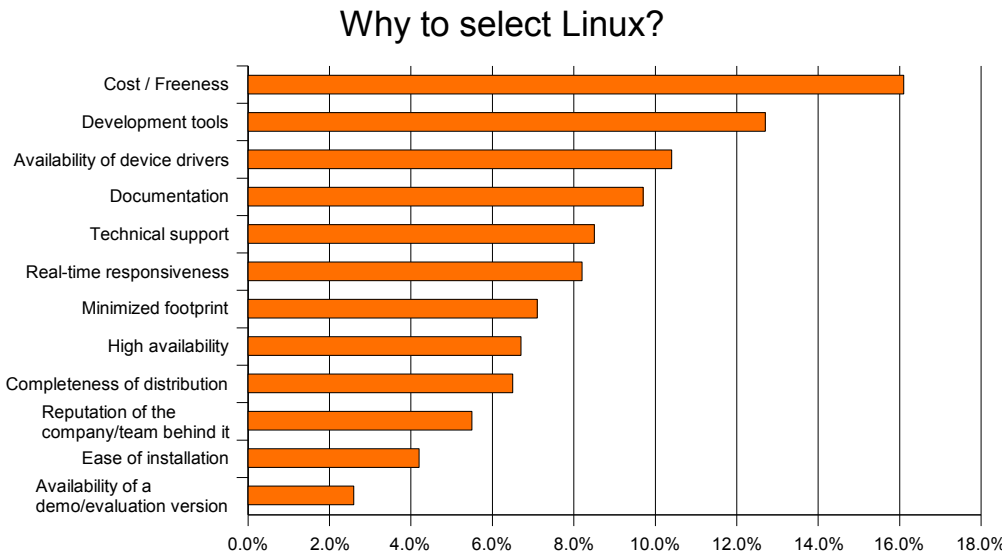


Figure 2. Influences for selecting Linux as the embedded operating system

2.4 Embedded Linux distributions

Unlike proprietary OSes, Linux is not controlled by any single authority (not even Linus Torvalds) who would decide its future and adoption of some technology over another. This gives to many companies and groups a chance to be part of the embedded Linux business. There are many vendors making their own embedded Linux distributions, and some of the most common ones are listed in following chapters.

2.4.1 Commercial distributions

Wind River

Wind River was founded in 1981, and it is headquartered in California, with operations worldwide. Wind River products and services are employed across a number of markets, including aerospace and defense, automotive, digital consumer, industrial, and network infrastructure.

Wind River is the leading supplier of real time operating systems (RTOS) and has just recently started to support Linux. In October 2005, Wind River launched a version of its commercial embedded Linux distribution targeting mobile phones, set-top boxes, PVRs (personal video recorders), and other small-footprint consumer electronics devices /38/.

MontaVista

MontaVista was founded in 1999 by Jim Ready, an embedded industry veteran. MontaVista powers the embedded revolution by providing GNU/Linux-based open-source software solutions. Its main product is MontaVista Linux, which is available in three versions: Professional (Pro), Consumer Electronics Edition (CEE) and Carrier Grade Edition (CGE). Professional is a general embedded OS featuring real-time functionality multi-process and multi-threaded, CEE is development environment for networked devices such as mobile phones, automotive telematics and high definition televisions. CGE is an industry standard Carrier Grade Linux providing functionality specifically for Telecom and Datacom with high availability, hardening and realtime performance. Motorola is using MontaVista in its recently launched Linux phones /14/.

LynuxWorks

LynuxWorks used to be known as Lynx Real-Time Systems, but decided then to embrace Linux and changed its name to reflect that decision. They still distribute and support their old proprietary Lynx operating system for hard real-time needs, and offer their embedded Linux distribution BlueCat for more powerful embedded usage. LynuxWorks was the first company to announce embedded Linux distribution based on the latest stable 2.6 kernel /15/.

Lineo Embedix

Lineo, Inc. provides embedded systems, real-time and high availability solutions that include software, hardware reference designs and professional services. At the end of 2002 Lineo Inc was acquired by Metrowerks, which is owned by Motorola. Metrowerks provides wide variety of Linux solutions, from optimized Kernel and platform to SDK for developers /16/.

2.4.2 Free distributions

Besides the commercial distributions presented in earlier section, open-source community has also free ones to select from. This list is even longer than the list of commercial distributors because there exists many 'one-man-projects' which are mainly used only by small community around their maintainer. List below includes only distributions with a bit wider user base.

Familiar
The Familiar Project is composed of a group of loosely knit developers all contributing to creating the next generation of PDA OS. It is a project started by Compaq, to offer open operating system for its iPAQ PDA series. Nowadays it's maintained and developed by community and contains multiple applications running on top of the distribution.

Starting from the version 0.8.0, Familiar is based on OpenEmbedded platform and offers two separate GUI systems: GPE which based on GTK+ toolkit and Opie which based on Qtopia toolkit /17/ /18/.

OpenEmbedded
OpenEmbedded is a full-featured development environment allowing users to target a wide variety of devices. Supporting multiple builds, release paths and configurations, OpenEmbedded extends the capabilities of build and release engineers. OpenEmbedded uses compilation and configuration caching at most levels to increase developer productivity /19/.

emdebian
Embedded Debian is a project to make Debian GNU/Linux a mainstream choice for embedded projects. Debian's multi architecture support, vendor independence, social contract and huge software base makes it an attractive choice for all sorts of systems, but the main distribution is very much aimed at systems with at least desktop resources. Embedded Debian tries to strip Debian down to be a much smaller system whilst keeping all the good things. Embedded Debian is currently very much a work-in-progress: plenty of people are already using Debian in their devices and systems, but there is huge potential to make this easier /20/.

DeviceLinux

DeviceLinux is an open source community focused on Linux-based software platforms. The community distributes and supports DeviceLinux Platform, a fully-fledged, license-free development environment with integrated development tools. It is mainly designed for manufacturers of network-enabled terminals and customizable for both consumer and industrial use. DeviceLinux works in tight cooperation with maemo.org and builds solutions on top of it /21/.

2.4.3 Linux from scratch

Building a self-made Linux distribution is relatively common among companies, especially in embedded devices. As the needs for the operating system in these devices vary greatly, it's usually quite natural to create own platform which fills the needs. Research among embedded Linux developers show that 'Home grown' distribution is the most common choice (19%) followed closely by ready made distributions /13/.

When building own Linux OS, it can be based on some already available distribution or building can be started from scratch. First option is more common as it is much easier road, using already working platform with some own additions and removing the unneeded features. This is the typical way how new Linux distributions have started up.

3 DEVELOPING SOFTWARE FOR SMALL DEVICES

This chapter introduces the aspects of mobile application development. Cross-compilation and usage of Scratchbox tool for this is also explained later in the chapter.

3.1 Application development, mobile aspects

Software development is a continuing process to create and maintain applications. There is no unambiguous definition for this process, because every application developed is a process on its own. Application development for embedded devices differ quite much from traditional application development done for desktop computers as the target for application is some smaller device like PDA, mobile phone or some other embedded platform. The main problem is that software development is quite resource hungry, the more calculation power system has the better. Because of this aspect, mobile development must be done in different machines than what the actual target is.

When normal PC applications can usually be developed and tested on the same system, embedded applications must be installed to actual device for testing purposes. This makes the development process much slower and has lead to different kinds of emulators which can be used in initial testing. These emulators have usually more or less shortcomings and final testing must still be done in real devices. Need for device testing grows when implementing lower level modules which are directly connected to hardware.

The mobile application development process can be accelerated greatly with the right tools and configurations. That is why several companies are selling SDKs specifically designed for embedded development. These development kits integrate usually the whole process and can be also quite expensive. All the tools needed for embedded Linux development are available freely, but setting them up requires lot of time and experience /37/.

Next section describes one important aspect of embedded development, Cross-compilation.

3.2 Cross-compilation

Cross-compilation is the process of compiling software for a CPU architecture different from the one used on the machine where the compilation is done. In Linux platform, this is normally accomplished by using a cross-compiler toolchain and cross-compiled libraries. These are then specified in the Makefiles that build the software.

This is relatively easy task with the low-level software and compilation tools, because they are usually designed to support cross-compilation. It's a very different matter with higher level software in general, especially with end-user applications and e.g. GUI toolkits /22/.

3.2.1 Problems

Most of the Linux open source software use GNU tools for building the application binaries from their source codes. This includes use of a 'configure' script to configure their software for compilation which on the other hand is produced by tools called 'autoconf' and 'automake'. These tools process the 'm4' macros written by the application developer. The script will generate the Makefiles that are used for building the software and a 'config.h' header file containing defines for features found on the build system /22/.

Configure is meant to ease configuring the software for compilation and its default assumption is that the software will be run in the same environment which it was compiled in and run from the place where it was installed to. This is exactly what is needed when normal desktop software is built, but it fails miserably when doing cross-compilation as the whole meaning of cross-compilation is that the target platform is different than the development platform.

Autoconf provides application developers certain macros to check out features in the system. The problematic macros for cross-compilation are:

- *AC_TRY_RUN* – Tries to compile, link and run given test code for some feature.

- *AC_TRY_LINK* – Tries to compile and link test code for library function existence.
- *AC_CHECK_LIB* – Tries to compile and link test code using certain library.
- *AC_SEARCH_LIBS* – Does the same as *AC_TRY_LINK*, but 'configure' tries to do linking from all of the system library paths, not just from ones given in *CPPFLAGS*, *CFLAGS* and *CXXFLAGS* environment variables.

These macros don't work in normal cross-compilation platform, so the whole build process fails /23/.

The second problem is that most application developers don't take cross-compilation into account when using these macros, so 'configure' ends up cross-compiling test code and trying to run it on the build host, which breaks the configuration. Configure can also find libraries, headers and versions of those that are only present on the build host, not on the target, which will fail either compilation of the program or running it on the target.

Third problem is when the developer does not know whether 'configure' found the correct values or not, unless he manually goes through tens of thousands lines of output it produces or when the binary fails when it's run. This results in a lot of manual work /23/.

3.2.2 Solutions

When there is a problem, usually also some solutions exist. There are mainly three ways for making GNU build tools work as intended when compiling to mobile devices.

- Select suitable software set and cross-compile them manually one-by-one.
- Compile Software natively, inside target device platform. This means that whole build process is done inside target device, or comparable device with same processor family and other configurations.
- Use special build environment for building the software. With this method configure scripts are 'faked' to think the build process happens in a different platform than it actually does.

First solution has been the most used one among developers, at least for slower target platforms. Its main problem is the manual editing it requires for each compilation to succeed which doesn't suit well to open source world. For this reason the platform is kept mainly static and newer reasons for various libraries and applications are not updated too frequently. It's just too hard to maintain a complicated system this way.

Second solution, building on a real embedded device, is also not very optimal. It does not scale and embedded devices, even developer editions, are always much slower in compiling than developer’s workstations. These devices also have space constraints which limit the development. There are special devices with overclocked mobile processor and memory that decrease compilation times, but they can be fairly expensive.

Scratchbox which is introduced in the next chapter is an example of third possibility. It is a build environment for cross-compiling software to different platforms and can be used in general x86 PC workstations.

3.3 Scratchbox

Scratchbox is a sandbox build environment for cross-compiling software. The sandbox contains only libraries and their includes / configuration utilities which are available on the target device. This helps to compile the software exactly as the target platform requires, with the same libraries installed to their directories.

3.3.1 CPU-transparency

CPU-transparency is the method Scratchbox uses to use host processor and target processor transparently when needed. This is automatically used in build process to get required information of compilation target platform. Also when running executable binary inside Scratchbox, it is run in host or target CPU depending on which target it's compiled for.

Scratchbox usage for CPU-transparency is explained in Figure 3.

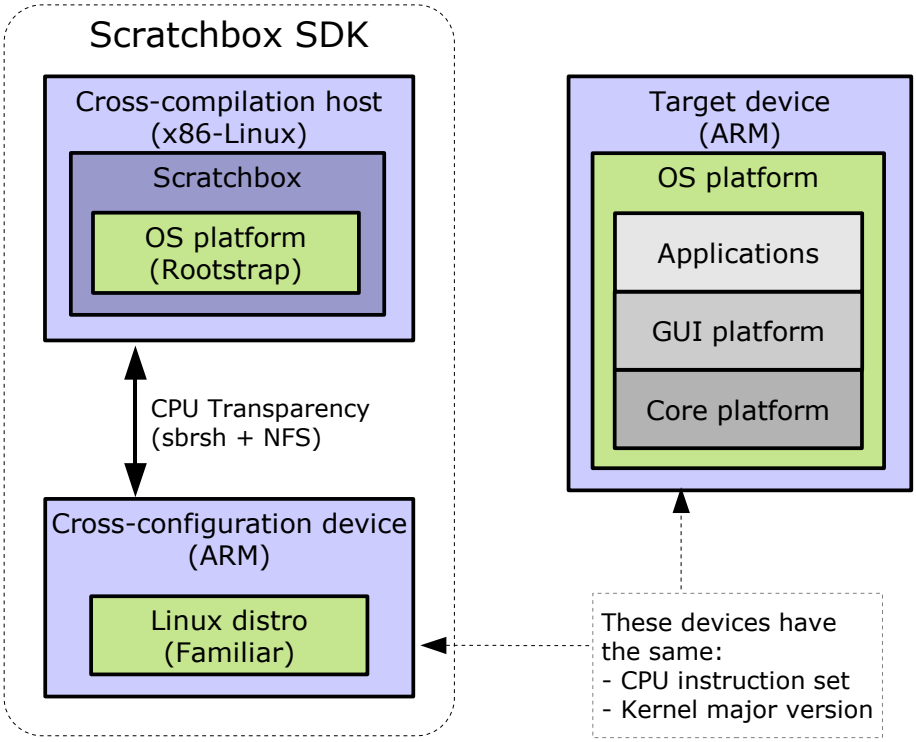


Figure 3. Scratchbox CPU transparency environment

Making this kind of environment between two very different platforms is not simple, because it must be so generic that it works with large set of already made applications. There are two important systems that Scratchbox uses to make cross-compilation work.

Fakeroot is a Debian utility used for building packages without real root privileges. It uses a daemon that keeps track of changes made to filesystems within the fake root environment. The communication is implemented with message queues and semaphores. Fakeroot-net is a special version of fakeroot developed for Scratchbox which uses TCP sockets for communication.

Sbrsh (Scratchbox Remote Shell) is a remote execution system that is used to implement CPU-transparency. It takes care of mounting network filesystems from host to target, creating a sandbox using chroot, copying environment variables, setting resource limits, etc. Sbrsh is also used in co-operation with fakeroot-net to make CPU-transparency work in a fakeroot environment.

The basic setting of Scratchbox system is shown in Figure 4.

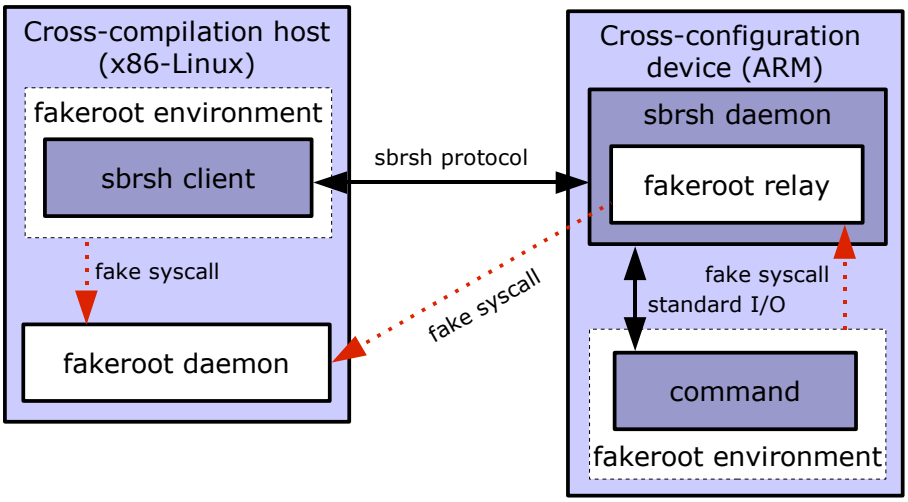


Figure 4. Scratchbox system with fakeroot

When logging into Scratchbox in the ARM-mode, it behaves exactly like a real Linux distribution installed on ARM device. This makes it possible to compile and install software so that everything can be installed on their default locations but nothing from the environment will be overwritten. The root filesystem inside Scratchbox should be made to match as close as possible the target root filesystem. Using the same versions of libraries and filesystem structure ensures that resulting binaries work also in the final target. Development tools and libraries that are needed to build software must be installed inside Scratchbox, but they are not needed in the final target device installation.

Scratchbox was used to cross-compile software used in this thesis, but its internals and usage are not explained here mode detailed. For more information about Scratchbox, see /24/.

4 GTK+ TOOLKIT

Second main subject of this master thesis is to concentrate on GTK+ toolkit and how it fits in embedded Linux platform. This chapter introduces all GTK+ related libraries and explains also how the GTK+ UI gets themed.

4.1 History

In the early days of the X Window System X11 the only popular toolkit was Motif. There were other toolkits available too, but they did not gain popularity because of different reasons. So when software developers Spencer Kimball and Peter Mattis decided to write an image-processing program in 1995, they used Motif toolkit.

This program (later to be called The GIMP) was distributed as free software and gained popularity fast. However, Motif was a commercial library so gaining wider user base was not easy. Therefore, Kimball and Mattis decided to write their own free toolkit and GTK (The GIMP Toolkit) was born /2/.

At the beginning GTK had three library components: Glib as a fundamental library, GDK as an interface to X11, and GTK on top of these. Somewhere on the line, GTK acquired object-oriented capabilities. Graphical components could now inherit from others, and the basic signal system that is still used was introduced. In honor of the new object-oriented features, developers decided to rename the toolkit to GTK+ /25/.

In GTK+ version 2.0, which was released on March 2002, the object-oriented pieces were separated to GObject system. In addition, GTK+ became platform independent by adding more back ends to GDK. Two more components appeared, Pango, a powerful library for text rendering, and ATK, an accessibility toolkit. At that point, GTK+ had no reason to be shy in comparison to any other toolkit /2/.

GTK+ 2.4 released on March 2004 added lots of improvements such as long waited new file dialog, action based toolbar and menu APIs, and Unicode 4.0 support. GTK+ 2.6 at the end of 2004 then came with new icon view widget, changes in file chooser and combobox widgets, improved clipboard support and

many performance improvements. This thesis is based on latest stable GTK+ 2.6 version and includes widgets which have only appeared in that version.

4.2 Introduction to GTK+

GTK+ is a multi-platform toolkit for creating graphical user interfaces. It offers very complete set of widgets to use when building applications. As described in earlier sentence, GTK+ was initially developed for and used by The GIMP, The GNU Image Manipulation Program. Today GTK+ is used by a large number of applications, and is the toolkit used by the GNU project's GNOME desktop /26/.

Figure 5 shows the location of GTK+ in normal Linux graphic pipeline, using Gnome desktop environment. These and other GTK+ related libraries are introduced in the next chapter.

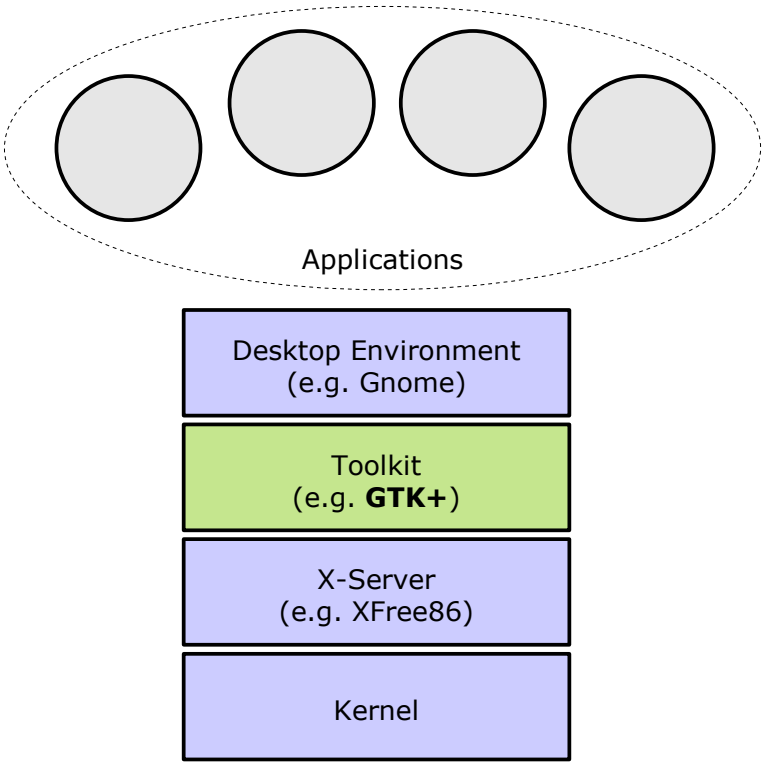


Figure 5. Position of GTK+ in graphical Linux system

4.3 GTK+ related libraries

GTK+ is built around strong basement of libraries which helps to modularize its structure. The different pieces of this structure are shown in Figure 6.

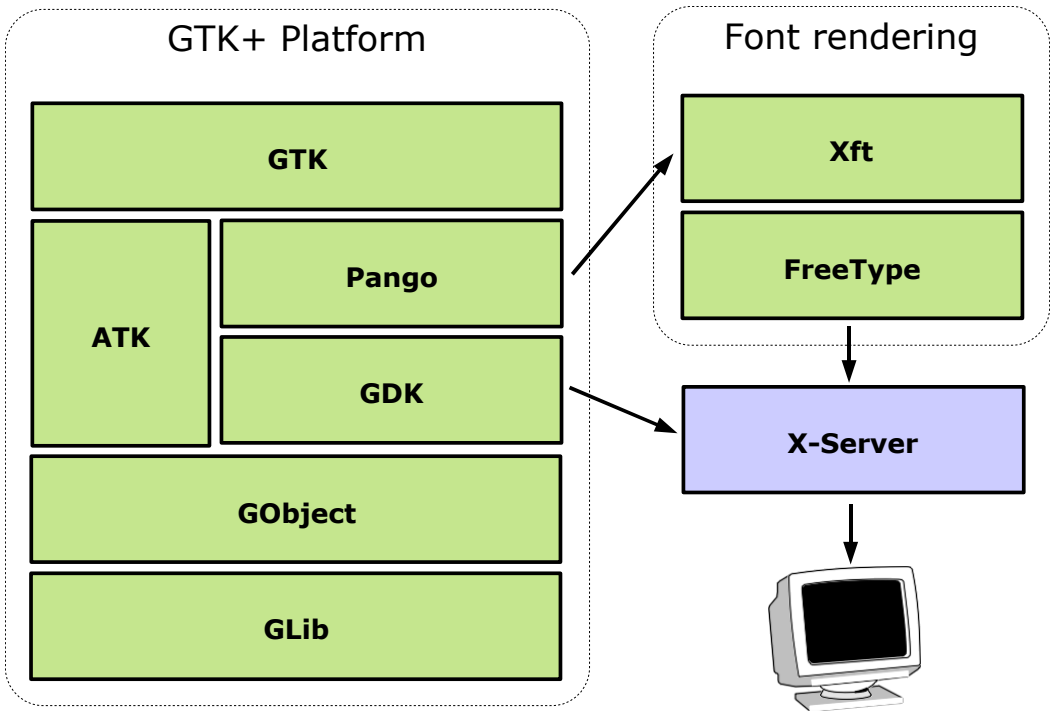


Figure 6. GTK+ platform libraries

GLib	Lower-level library that provides many useful definitions and functions for GTK+ applications.
GObject	The object system whole GTK+ is based on.
GDK	Library which provides a graphical layer and functions to X-server.
Pango	Framework for the layout and rendering of internationalized text.
ATK	Library provides a set of interfaces for supporting different accessibility features.
Xft	Provides a client-side font API for X applications. Uses Fontconfig to select fonts and the X protocol for rendering them.
FreeType	Font rendering library which renders font character bitmaps.
X-Server	Graphical server which provides X11 compatible interface to display device.

These libraries inside GTK+ platform are next described in more details. The font rendering parts are not major part of this thesis, so information about them can be found for example from /27/.

4.3.1 GLib

GLib is a lower-level library that provides many useful definitions and functions available for use when creating GDK and GTK+ applications. These include definitions for basic types and their limits, standard macros, type conversions, memory allocations, warnings and assertions, message logging, timers, string utilities and other usable features.

To use GLib and all its types, glib.h header file needs to be included in source codes:

```
#include <glib.h>
```

After this line all GLib functionality can be used. GLib is used by GTK+ but not another way around, so non-GTK+ applications can and do use GLib as well. GLib includes number of utilities that simplify everyday interaction with the C programming language and the system:

- **Data structures** – GLib has number of standard implementations for common data structures, including lists, trees and hash tables. These make programming easier and less error prone, as application makers don't have to recreate these frequently needed data structures themselves.
- **Memory management** – GLib applications should use memory management routines provided by it instead of standard C programming language *malloc()*, *realloc()*, and *free()*. The advantage of these GLib functions is the build-in error handling.
- **Unicode and character encodings** – The traditional C string functions use byte strings and don't have to worry about the length of an individual characters because each of them is one byte long. Because GTK+ tries to be fully internationalized it uses Unicode characters. GLib generally uses the

32-bit UCS-4 type as its internal Unicode standard and provides functionality to encode text to other character modes.

- **Logging** – To help runtime error diagnosis, GLib offers several utilities for logging messages to system console and log files.
- **Debugging and exception handling** – GLib has several facilities which help to locate the bugs in applications. These are heavily used in other parts of GTK+.
- **Shell and file utilities** – GLib contains number of functions for working with files, pipes, and processes.

GLib is used to make applications portable across operating systems (e.g. Linux and Windows) and hardware platforms (e.g. X86 and PowerPC). For this reason it is important to use Glib basic variable types instead of C types in GTK+ applications. Table 1 lists most notable ones of them, for a full list see /2/.

Table 1. Most important GLib basic types.

GLib Type	*Corresponding type in ANSI C*
gchar	char
guchar	unsigned char
gint	int
guint	unsigned int
gfloat	float
gdouble	double
gpointer	void *, untyped pointer
gboolean	Boolean value, TRUE or FALSE

Following application shows some basic concepts of GLib GList usage.

```
#include <glib.h>

/* Print out the value from list */
void print_number(gpointer data_ptr, gpointer ignored)
{
  g_print("%d ", *(gint *)data_ptr);
}

int main(int argc, char **argv)
```

```
{
  GList *list = NULL;
  gint i;
  gint *data_ptr;

  g_print("\nInitializing list...");
  for (i=0; i<10; i++)
  {
     data_ptr = g_new(gint, 1);
     *data_ptr = i;
     list = g_list_append(list, (gpointer)data_ptr);
  }

  /* Print out the content of list */
  g_print("\nList of %d items: ", g_list_length (list));
  g_list_foreach(list, print_number, NULL);

  /* Reverse list content */
  list = g_list_reverse(list);
  g_print("\nList reversed: ");
  g_list_foreach(list, print_number, NULL);

  /* Add the meaning of life at beginning of list */
  data_ptr = g_new(gint, 1);
  *data_ptr = 42;
  list = g_list_prepend(list, (gpointer)data_ptr);
  g_print("\nList of %d items: ", g_list_length (list));
  g_list_foreach(list, print_number, NULL);

  /* Print third (=2) item of list */
  data_ptr = g_list_nth_data(list, 2);
  g_print("\nThird item: %d \n", *data_ptr);

  g_list_free(list);
  return 0;
}
```

Application creates list structure with content, reverses it and adds some more items. Output of the application is below.

```
kaj@ubu:~ $ ./glib_example
Initializing list...
List of 10 items: 0 1 2 3 4 5 6 7 8 9
List reversed: 9 8 7 6 5 4 3 2 1 0
List of 11 items: 42 9 8 7 6 5 4 3 2 1 0
Third item: 8
```

This example only shows very small part of GLib functionality and there are also many minor facilities which have not been listed here at all. But unfortunately there is no possibility to proceed deeper here, for more information about GLib see the official GLib tutorial /28/.

4.3.2 GObject

Most modern programming languages come with their own native object systems and additional fundamental algorithmic language constructs. Just as GLib serves as an implementation of such fundamental types and algorithms, GObject (GLib Object System) provides the required implementations of an extensible object-oriented framework for non-object-oriented C-language. It supports all basic object-orientation basics with classes, inheritance and interfaces, which are required for modern GUI widget system such as GTK+.

With GObject more work is required to accomplish this than for example with Java or Python, just because C-language is not originally designed to be object-oriented. This is easiest to describe with a small example of making classes from GObject, such as structured in Figure 7.

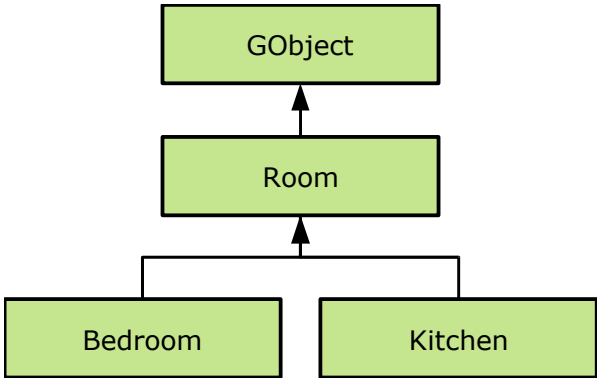


Figure 7. Object hierarchy of house area

Code below shows how to define *Room* as instance structure and *RoomClass* as the class structure using GObject.

```
/* Instance structure */
typedef struct _Room
{
  GObject parent_instance;

  guint area;
  GString *title;
  gboolean electricity;
} Room;

/* Class structure */
typedef struct _RoomClass
{
  GObject parent_class;

  /* Signals */
  void (*electricity_added)(Room *room);
} RoomClass;
```

The attributes in the instance structure include *area*, *title* and *electricity*. These represent room size, type and whether it has electricity, respectively. The class structure includes also handler prototype for one signal: *electricity_added*. This signal is followed when Room becomes electricity attached.

Following utility macros are required for all GObject classes. Note the usage of 'klass' instead of 'class', as the second one is a keyword in C++ and GObject is designed to be C++ compatible.

```
#define TYPE_ROOM(room_get_type())
#define ROOM(object)
    (G_TYPE_CHECK_INSTANCE_CAST((object),
    TYPE_ROOM, Room))
#define ROOM_CLASS(klass)
    (G_TYPE_CHECK_CLASS_CAST((klass),
    TYPE_ROOM, RoomClass))
#define IS_ROOM(object)
    (G_TYPE_CHECK_INSTANCE_TYPE((object),
    TYPE_ROOM))
#define IS_ROOM_CLASS(klass)
    (G_TYPE_CHECK_CLASS_TYPE((klass),
    TYPE_ROOM))
#define ROOM_GET_CLASS(object)
    (G_TYPE_INSTANCE_GET_CLASS((object),
    TYPE_ROOM, RoomClass))
```

Explaining these macros one by one:

- TYPE_ROOM returns the class type identified, a GType assignment for the Room class. This macro is used in any places that calls for an object or type identifier.
- ROOM() is a casting macro that casts an object to the Room type.
- ROOM_CLASS() is another casting macro which operates on the class structure instead, returning a casted RoomClass structure.
- IS_ROOM() checks whether object belongs to to the Room structure.
- IS_ROOM_CLASS() checks the membership of a class for the RoomClass structure.
- ROOM_GET_CLASS() yields the class structure of Room when an instance of the Room structure.

When these general macros are introduced, class can be improved with methods to provide real functionality to it. Next code block presents two example methods, first which prints out room information and second which is used to set the room area.

```
/* Class methods */
void room_print_info(Room *object)
{
  Room *room;
  g_return_if_fail(IS_ROOM(object));
  room = ROOM(object);
  g_print(“Room area: %d”, room->area);
}

room_set_area(Room *object, guint new_area)
{
  Room *room;
  g_return_if_fail(IS_ROOM(object));
  room = ROOM(object);
  room->area = new_area;
}
```

Signals are events that can happen to object during its life. They serve as a means of communication between objects, when receiving signal the object can react on it in proper way. The signal system in GTK+/GObject is called GSignal. As an example of attaching signals to GObject, next code block adds listener to signal which gets handled when room gets attached with electricity.

```
/* Signal indices */
enum {
  ELECTRICITY_ADDED
}

/* Install signal */
room_signal[ELECTRICITY_ADDED] =
  g_signal_new(“electricity_added”,
```

```
      TYPE_ROOM,
      G_SIGNAL_RUN_LAST|G_SIGNAL_DETAILED,
      G_STRUCT_OFFSET(RoomClass, electricity_added),
      NULL,
      NULL,
      g_cclosure_marshal_VOID__VOID,
      G_TYPE_NONE,
      0);

/* Signal handler */
static void room_electricity_attached(Room *room)
{
  if (!room->electricity)
  {
    g_object_set(room, “electricity”, TRUE, NULL);
    g_print(“Room got electricity!\n”);
  } else {
    g_print(“Room already has electricity!\n”);
  }
}
```

There is a lot more in GObject which has not been explained here, for more information see /21/ and /2/.

4.3.3 Pango

The Pango libraries provide an open-source framework for the layout and rendering of internationalized text. Pango is closely related to GTK+ and GNOME projects, and in fact it was originally developed only to GTK+. However there is nothing fundamentally GTK+ or GNOME specific about Pango, and it can also be used without them.

Pango uses Unicode standard for all of its encoding, which helps it to support output practically in all the worlds major languages. Pango toolkit provides basically simple string drawing rendering calls and text entry and label widgets that transparently support Unicode and multi-lingual text.

Layout and rendering in Pango involves several steps /29/:

- **Itemization** – The input string is broken into portions rendered with a consistent font, language tag, and a specific bidirectional embedding level.
- **Reordering** – The items are reordered from logical order into visual order according to their bidirectional embedding levels.
- **Glyph Selection** (Shaping) – The characters in each item are turned into glyphs.
- **Justification** – The glyph strings created in the previous step are adjusted to fit the line-justification policies that are in place.
- **Rendering** – The justified glyph strings are rendered in their final order onto the output device. Pango itself is not really responsible of this step, in Linux/GTK+ system font rendering is handled usually by FreeType.

Figure 8 shows Gnome sound preferences dialog with English and with Hebrew locales. As visible from there, changing the locale doesn't just change the language used in text labels, but also the whole arrangement of the interface. As the Hebrew is written from right to left, all positioning is changed. The power of Pango here is that this all happens automatically and software designer doesn't need to do anything else than provide the localization strings, Pango does all the rest.

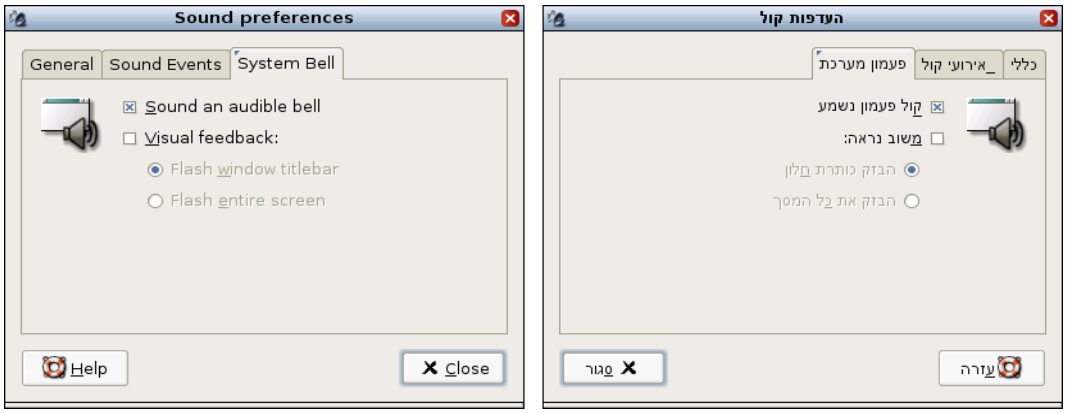


Figure 8. GNOME Sound preferences dialog with two different locales.

4.3.4 GDK

GDK (GTK+ Drawing Toolkit) is a layer which provides all drawing primitives for GTK+. It is mostly only a wrapper for low-level windowing functions, but includes some extra functionality too. There are mainly two reasons for GDK to exist:

- GDK adds layer on top of actual drawing functionality of the platform and this way offers compatibility between Linux (X11) and Windows (GDI). Same way as GLib makes it easier to port applications to different platforms, GTK+ user interface can be ported by making a GDK wrapper on top of platforms own drawing functions.
- GDK makes drawing functionality to support GLib and GObject, providing drawing primitives which are easier to use from GTK+ than normal X11 drawing functions.

Source code below demonstrates basic capabilities of GDK, drawing lines, ovals, boxes and images.

```
GdkPixmap *pixmap = NULL;

void draw(GtkWidget *widget)
{
    /* Set rectangle area which will be updated */
    GdkRectangle update_rect;
    update_rect.x=0;
    update_rect.y=0;
    update_rect.width=widget->allocation.width;
    update_rect.height=widget->allocation.height;

    /* Draw white rectangle to whole area */
    gdk_draw_rectangle(pixmap,
        widget->style->white_gc,
        TRUE,
        0,0,
        widget->allocation.width,
```

```
        widget->allocation.height);

    /* Draw line */
    gdk_draw_line (pixmap,
        widget->style->black_gc,
        10, 10, 30, 90);

    /* Draw filled circle */
    gdk_draw_arc (pixmap,
        widget->style->black_gc, TRUE,
        60, 10, 50, 80, 0, 360*64 );

    /* Draw unfilled rectangle */
    gdk_draw_rectangle(pixmap,
        widget->style->black_gc, FALSE,
        140, 10, 40, 80);

    /* Draw pixmap image */
    GdkPixbuf *pixbuf_drawing =
        gdk_pixbuf_new_from_file("terminal.png", NULL);
    gdk_draw_pixbuf (pixmap,
        NULL, pixbuf_drawing,
        0, 0, 200, 5, -1, -1,
        GDK_RGB_DITHER_NONE,
        0, 0);

    /* Update drawing area */
    gtk_widget_draw(widget,&update_rect);
}
```

These GDK drawing functions are quite self-explanatory, even thought object-orientation makes a small extra work in them. Figure 9 shows a screenshot of this example application.

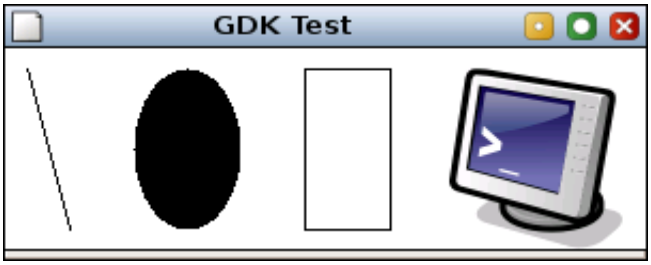


Figure 9. GDK drawing example

GDK is important for GTK+ in the conceptual level, but for this thesis it's just good to know that this layer exists when more advanced modifications and graphical features are needed.

4.3.5 ATK

The ATK library provides a set of interfaces for supporting different accessibility features. By supporting the ATK interfaces, an application or toolkit can be used with such tools as screen readers, magnifiers, and alternative input devices.

Most of the general accessibility requirements are already implemented in GTK+ widgets, so they are transparent to application developers. But when own special widgets are needed to be developed, ATK interfaces should be used to meet the accessibility needs. This is however out of the scope of this thesis, to learn more about ATK see /30/ and /26/.

4.4 GTK+ language bindings

GTK+ library is programmed with C-language, which makes C the natural programming language to build GTK+ applications with. But GTK+ was also designed from the very beginning to support other programming languages. These language bindings (also called 'wrappers'), allow GTK+ to be used from other programming languages, in the style of those languages.

There have been support for many languages in the history of GTK+, but not all of them are actively maintained and support the latest versions. Table 2 lists languages supported by Gnome Language Bindings /26/.

Table 2. Languages supporting Gnome Platform Bindings.

Language	*GTK+ 2.0*	*GTK+ 2.2*	*GTK+ 2.4*	*GTK+ 2.6*
C++	complete	complete	complete	complete
Java	supported	supported	complete	complete
Perl	supported	supported	complete	complete
Python	complete	complete	complete	complete

GTK+ bindings are available for several other programming languages too, but most of them support only part of the GTK+ API or older GTK+ versions. It is relatively safe to say that from the list above, C++ is most practical for embedded usage due to performance reasons. All other languages are not compiled to binary, but need a separate runtime environment which makes them perform slower.

4.5 GTK+ theming system

Theming is the ability to provide different appearances for the widgets which build the user interface of applications. With a rich theming system, it is possible to change the appearance and behavior of the widgets without any code modifications. This can also help to suit better the needs of mobile devices as. the small screen resolution and touch screen input method of these devices require different kind of theme than a normal desktop PC.

The theming ability in GTK+ differs somewhat from other toolkits in that a GTK+ theme doesn't define the appearance for entire widgets, but instead defines ways of drawing a number of different graphical elements, such as beveled boxes, frames, and check button indicators. This approach allows entirely new widgets that are composed with these same elements to draw correctly with the new theme.

A GTK+ theming consists of three elements:

- Theme engine, which is a shared object including code to draw the graphical elements. Different theme engines are described in chapter 4.5.1
- Configuration file to configure the operation of the theme engine and the core parts of GTK+. The structure of the GTK+ resource file (gtkrc) is described in chapter 4.5.2

- Data files for the theme engine such as images. Normally images are bitmap graphic files (PNG, GIF), but GTK+ supports also scalable vector graphics (SVG) which are mostly used for theme stock icons.

A theme can include its own theme engine, it can use an existing theme engine, or it can use GTK+'s built in drawing code. All aspects of the theme are configured on a per-widget basis and each widget contains a pointer to a *GtkStyle* object that defines the theme engine and theme options for that particular widget. So, in fact, a theme can apply different theme engines to different widgets. While using two separate external theme engines is not normal, it is pretty common to have a theme that uses an external theme engine for some widgets, and the default built-in GTK+ theme engine for other widgets that require less treatment.

Figure 10 shows the same GTK+ application with different themes attached.

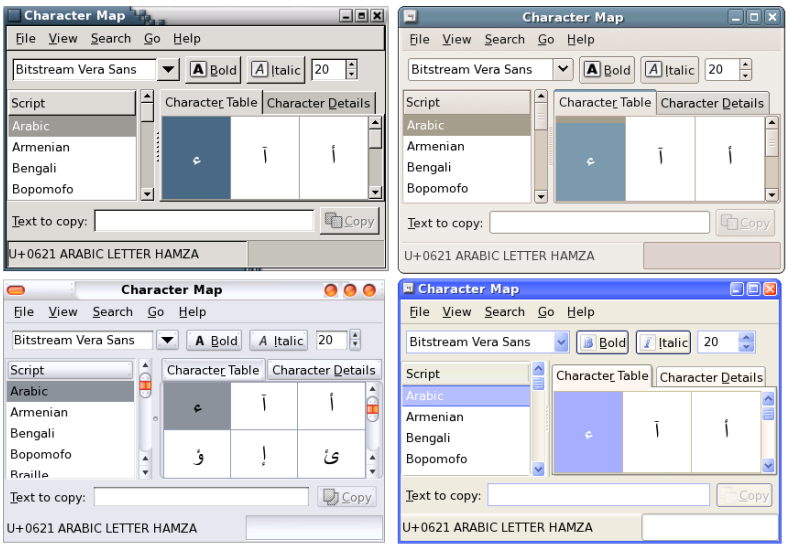


Figure 10. Different GTK+ themes

Next section describes theme engines which are used to extend the GTK+ theming ability.

4.5.1 Theme engines

A theme engine is the part of GTK+ which is responsible of drawing the widgets (like buttons, menus, scrollbars, progress bars etc.). There are two theme engines provided with the GTK+ itself, default drawing engine and pixmap engine (called 'pixbuf' after GTK+ 2.6) which uses images for all controls. This image usage makes themes based on pixmap engine very customizable, but also usually slower and memory consuming.

There are also external theme engines which all have their own specific look and are drawn using C code so they can be faster compared to just using images. The different themes available for these engines typically just change some colors and a few other parameters, like where to place scrollbar arrows. Some engines give quite a lot preferences for theme to adjust, some allow only general color modifications. Table 3 lists the most commonly used GTK+ theme engines.

Table 3. GTK+ theme engines

Theme engine	***Description***
Default	Default GTK+ engine is very simple and only drawing is used for widgets. This makes it fast but also quite plain looking.
Pixmap	Pixmap engine was first engine to use images for widget backgrounds. It's not commonly used anymore as more advanced themes have displaced it. With the release of GTK+ 2.6, the pixmap (now called 'pixbuf') theme engine has been integrated into the GTK+ sources.
Mist	Mist is a semi-flat theme engine with the intent to be simple and fast. It uses a combination of thin and flat edges to provide a rather unique look. It does not provide any configuration to change its appearance.
Xfce	Xfce engine is designed to be used with Xfce window manager, but it is normal GTK+ theme and can be used also with GNOME.
Smooth	Smooth theme engine is a simple theme engine intended to be smooth, fast, and highly configurable. It is trying to provide appealing interface while still retaining a small memory footprint.
Industrial	Industrial engine was designed by company called Ximian to provide modern GUI to their GNOME desktop.
Cleanice	The Cleanice Engine is a simple single look engine approach with no extra configurations. It aims towards good, clean and simple defaults, which is supporting the GNOME ideal that

	sane defaults are better than complete configuration. It has also a slightly modified cousin, Thinice.
eXperience	This engine is made to match Windows XP™ theme.

Engine is attached to theme style with the “engine” tag in GTK+ theme resource file.

```
style "default" {
    engine "pixmap" {
        ...
    }
}
```

The format of this gtkrc file is explained in the next section.

4.5.2 GTK+ resource files

GTK+ theming system is based on drawing engines which are configured with one or more GTK+ resource files (gtkrc). The resource file format is based on C-like block style syntax with declarations and bindings. Theme is built from a bunch of style declarations which are then bound to the specific widgets. The properties which are set in bound style declarations are passed down the widget inheritance tree unless overridden somewhere along the way /31/.

A style is defined by the keyword "style" followed by a name for the style and optionally another style name to inherit properties from. After the style has been defined, it can be attached to some widget class. Here's an example:

```
style "style-name" {
    # style definitions
}
widget_class "GtkMenuItem" style “style-name”
```

This defines the style named "style-name" and then applies it to GtkMenuItem widgets. Class can also be "*", which will attach the style then to all widget classes. There can be as many uniquely named styles in gtkrc file as are needed.

Resource files support also inheritance. To inherit another style's properties following structure can be used:

```
style "my-style-1" {
    # style definitions for style 1
}
style "my-style-2" = "my-style-1" {
    # style definitions, explicitly to style 2
}
```

The “my-style-2” will inherit the first style's properties, and properties defined explicitly in the second style will take precedence.

Colors are defined with “COLORFLAG[WIDGETSTATE] = COLOR” lines where:

- COLORFLAG is “bg” (background), “fg” (foreground), “text” (text color) or “base”.
- WIDGETSTATE is “NORMAL”, “ACTIVE”, “PRELIGHT”, “SELECTED” or “INSENSITIVE”.
- COLOR is a value of color described in one of the varying formations, for example #rrggbb or {r, g, b}.

Below is an example of defining colors to menu items in their PRELIGHT state.

```
style "menu-item" {
  bg[PRELIGHT]   = "#336699"
  fg[PRELIGHT]   = "#FFFFFF"
  base[PRELIGHT] = "#336699"
  text[PRELIGHT] = "#FFFFFF"
}
```

Widget style properties are public widget class properties which can be set in themes via a C++ like format. These properties include different ways to customize theme appearance by adjusting widget thickness, size, shadow style etc. Format of these style lines is “WIDGETCLASS::property_name = property_value” like in the following example:

```
GtkScrollbar::min_slider_length = 14
GtkCheckButton::indicator_size = 10
GtkCheckMenuItem::indicator_size = 10
GtkButton::default_border = { 0, 0, 0, 0 }
```

Full list of GTK+ resource elements and states are available in /32/.

5 GTK+ TOOLKIT WIDGETS

This chapter introduces GTK+ GUI widgets. Being one of the main aspects of this thesis, following chapter is the longest one even thought it contains only the most commonly used widgets. This is anyway important to present what GTK+ offers for the software developers and how to get full advantage of it. Full class hierarchy of GTK+ widgets is available in Appendix A.

5.1 Windows

Windows are usually the basis in all GUI applications, also in GTK+. *GtkWindow* is the top level window which contains all other widgets. *GtkWindows* with the default type are managed by the window manager and they include a frame for content by default, but it's also possible to get full control of the window events and decoration. Figure 11 shows an empty *GtkWindow* widget.

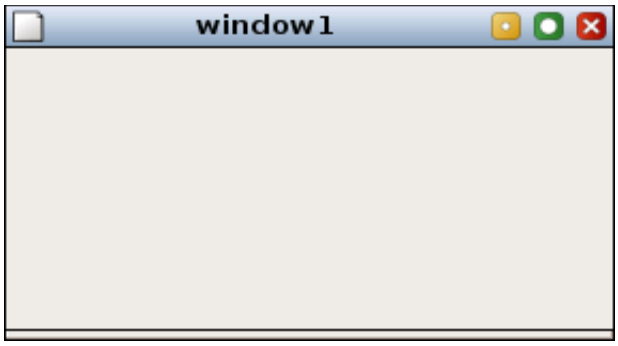


Figure 11. GtkWindow

GtkDialog is inherited from *GtkWindow*. Dialogs are popup windows which are a convenient way to prompt the user for a small amount of input, e.g. to display a message, ask a question, or anything else which does not require extensive effort on the user's part. Figure 12 presents *GtkDialog* with three default buttons: Help, Cancel and OK.

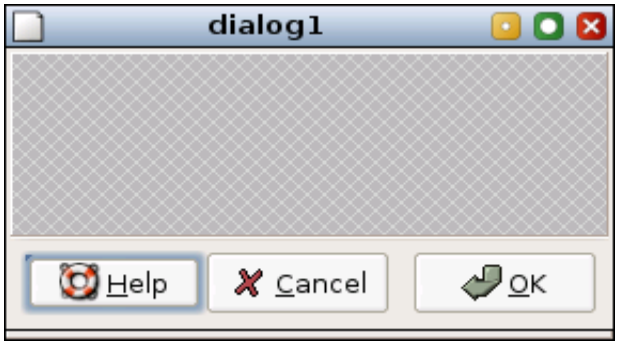


Figure 12. GtkDialog with 3 buttons

GtkDialog can be used to create custom dialogs, but there are also ready-made ones in GTK+, such as *GtkAboutDialog* to display application information and *GtkMessageDialog* which is more convenient way to show small notifications.

5.2 Selectors

Selectors are also dialogs, but they are designed to some definite purpose with limited customization possibilities.

GtkColorSelectionDialog is used to let the user select a specific color. It uses *GtkColorSelection* widget for the graphical selection area on the left side of the dialog (Figure 13) and can be launched also with *GtkColorButton* which is a special button showing the currently selected color.

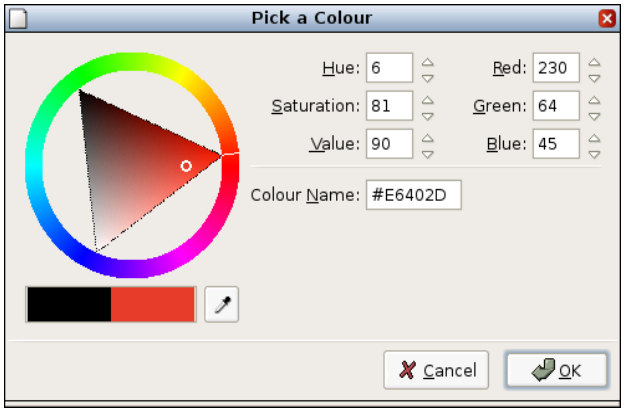


Figure 13. GtkColorSelectionDialog

GtkFontSelectionDialog is like the color selection dialog, but it's used to select the font and the font styles. It has also a convenience button (*GtkFontButton*) showing currently selected font name and size. Figure 14 presents this dialog.

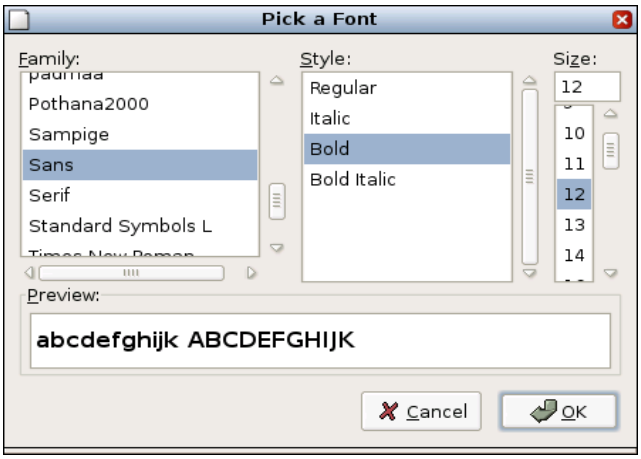


Figure 14. GtkFontSelectionDialog

GtkFileChooserDialog is used for file handling. It has two separate modes, one for saving files and another for opening them. As with the other selectors, also *GtkFileChooserDialog* has a *GtkFileChooserButton* to show selected file name and launch the main dialog.

Older GTK+ versions before 2.4 had a much criticized *GtkFileSelector* widget for file handling functionality, but luckily it is now deprecated and replaced with this more powerful one. There were a lot of background studying done of file dialogs in other operating systems before selecting the interface shown in Figure 15. Interface of the dialog is still improving with every new GTK+ releases.

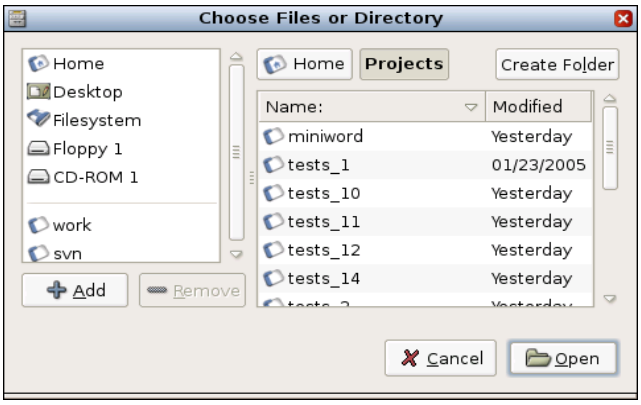


Figure 15. GtkFileChooserDialog in open mode

5.3 Layout containers

Layout containers are used (as the name describes) to layout the widgets in their own places to create the application user interface. Figure 16 demonstrates different containers used to arrange buttons inside the window. The source code used to create this user interface is sliced and explained during this chapter.

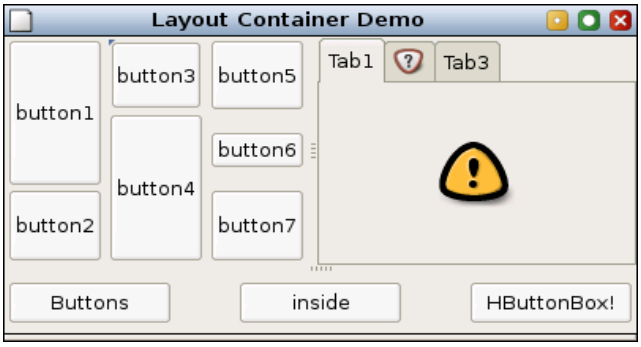


Figure 16. Different GTK+ layout containers

GtkHBox and *GtkVBox* are the most basic layout separators in GTK+. They are used to slice the remaining canvas space in one or more horizontal (*GtkHBox*) or vertical (*GtkVBox*) areas. *GtkBox* packing interface is then used to determine the arrangement, spacing, width, and alignment of widgets placed inside these areas. Example of *GtkBox* usage is presented below.

```
/* Create horizontal GtkHBox */
hbox1 = gtk_hbox_new (FALSE, 0);
gtk_widget_show (hbox1);
gtk_container_add (GTK_CONTAINER (window1), hbox1);

/* Create vertical GtkVBox with 2px spacing and
    4px borders*/
vbox1 = gtk_vbox_new (FALSE, 2);
gtk_widget_show (vbox1);
gtk_box_pack_start
    (GTK_BOX (hbox1), vbox1, TRUE, TRUE, 0);
gtk_container_set_border_width
    (GTK_CONTAINER (vbox1), 4);
```

The *GtkButtonBox* classes are like *GtkBox* widgets but they are specially designed to arrange buttons identically on line. Buttons added as the children's inside these containers will have the same height and width, so that buttons have a consistent layout. Source code below creates three buttons inside *GtkHButtonBox* as in bottom of Figure 16.

```
/* Create GtkHButtonBox with 2px border */
hbuttonbox1 = gtk_hbutton_box_new ();
gtk_paned_pack2 (GTK_PANED (vpaned1), hbuttonbox1,
    FALSE, FALSE);
gtk_container_set_border_width (GTK_CONTAINER
    (hbuttonbox1), 2);

/* Add three buttons */
hbutton1 = gtk_button_new_with_label("Buttons");
gtk_container_add (GTK_CONTAINER (hbuttonbox1),
    hbutton1);
hbutton2 = gtk_button_new_with_label("inside");
gtk_container_add (GTK_CONTAINER (hbuttonbox1),
    hbutton2);
hbutton3 = gtk_button_new_with_label("HButtonBox!");
gtk_container_add (GTK_CONTAINER (hbuttonbox1),
    hbutton3);
```

GtkHPaned and *GtkVPaned* differ from their *GtkBox* siblings mainly in two ways:

- These widgets separate canvas always to only two pieces. Multiple areas can't be created with one *GtkPaned*.
- Between these two areas becomes a handle separator which the user can drag to adjust the division. This is handy when content of both sides is changing and user is wanted to give the opportunity to decide which side he wants to keep more visible.

GtkPaned is the base class for widgets with two panes, arranged either horizontally (*GtkHPaned*) or vertically (*GtkVPaned*). Both children widgets inside *GtkPaned* have two options that can be set, *resize* and *shrink*. If *resize* is TRUE,

then when the *GtkPaned* is resized, that child will expand or shrink along with the paned widget. If *shrink* is TRUE, then that child can be made smaller than its requisition by the user. Source code below creates two paned views which separate different container types in Figure 16.

```
/* Create vertical GtkVPaned */
vpaned1 = gtk_vpaned_new ();
gtk_widget_show (vpaned1);
gtk_container_add (GTK_CONTAINER (window1), vpaned1);

/* Create horisontal GtkHPaned */
hpaned1 = gtk_hpaned_new ();
gtk_widget_show (hpaned1);
gtk_paned_pack1 (GTK_PANED (vpaned1), hpaned1,
    FALSE, TRUE);
```

GtkTable is a bit more complicated layout manager, at least when structuring it manually. Its functions allow the programmer to arrange widgets in rows and columns, making it easy to align many widgets next to each other, horizontally and vertically. But it has also more sophisticated methods to let child's expand, shrink and fill to both x and y coordinates. Source code below shows only part of the button table creation for Figure 16 as full source would be too long.

```
/* Create GtkTable */
table1 = gtk_table_new (3, 3, FALSE);
gtk_widget_show (table1);
gtk_container_set_border_width (
    GTK_CONTAINER (table1), 2);
gtk_table_set_row_spacings (GTK_TABLE (table1), 2);
gtk_table_set_col_spacings (GTK_TABLE (table1), 4);

/* Add 2 rows high button to top-left corner */
button1 = gtk_button_new_with_mnemonic (_("button1"));
gtk_table_attach (GTK_TABLE (table1), button1, 0, 1,
    0, 2, (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
    (GtkAttachOptions) (GTK_EXPAND | GTK_FILL), 0, 0);
```

```
/* Add 1 row high button to top-right corner */
button2 = gtk_button_new_with_mnemonic (_("button2"));
gtk_table_attach (GTK_TABLE (table1), button2, 0, 1,
    2, 3, (GtkAttachOptions) (GTK_FILL),
    (GtkAttachOptions) (GTK_EXPAND | GTK_FILL), 0, 0);
...
```

GTK+ includes also a tabbed notebook container, *GtkNoteBook.* Children of this container are pages which can be switched between by using the tabs labels along one edge. These tabs can be placed on all four sides of the container. As seen in Figure 16, tab handlers are not limited to text, but can contain images or almost any other GTK+ widgets. This is useful e.g. when each tab should contain a button to close the tab. The source code here creates first two tabs shown in Figure 16.

```
/* Create GtkNotebook */
notebook1 = gtk_notebook_new ();
gtk_paned_pack2 (GTK_PANED (hpaned1), notebook1,
    TRUE, TRUE);

/* Add content & label to first note */
image1 = gtk_image_new_from_stock (
    "gtk-dialog-warning", GTK_ICON_SIZE_DIALOG);
gtk_container_add (GTK_CONTAINER (notebook1), image1);
label1 = gtk_label_new (_("Tab1"));
gtk_notebook_set_tab_label (GTK_NOTEBOOK (notebook1),
gtk_notebook_get_nth_page (
    GTK_NOTEBOOK (notebook1), 0), label1);

/* Add content & image to second note tab */
empty_notebook_page = gtk_vbox_new (FALSE, 0);
gtk_container_add (
    GTK_CONTAINER (notebook1), empty_notebook_page);
image2 = gtk_image_new_from_stock (
    "gtk-dialog-question", GTK_ICON_SIZE_BUTTON);
```

```
gtk_notebook_set_tab_label (GTK_NOTEBOOK (notebook1),
gtk_notebook_get_nth_page (
    GTK_NOTEBOOK (notebook1), 1), image2);
...
```

GTK+ includes also a container which allows to position widgets at fixed coordinates. Inside *GtkFixed,* locations are given in pixels, from top-left corner. This layout type should be avoided as it doesn't adjust to different window sizes, themes which may change widget sizes, fonts other than the one used to write the application, or even translated text stings.

5.4 Display widgets

Display widgets are widgets which are used to show information to user, but can't listen events and so on are non-interactive as default. Display widgets are usually placed inside containers presented in earlier chapter. Figure 17 presents three most used display widgets: *GtkImage*, *GtkLabel* and *GtkStatusbar*.

Figure 17. GtkImage, GtkLabel and GtkStatusbar

GtkImage widget is used to display images. The image file may also contain multiple images. If so, the *GtkImage* will automatically display an animation (*GdkPixbufAnimation*) instead of a static image. *GtkImage* does not receive events by default and if events such as button clicks are wanted to be received, the widget should be placed inside a *GtkEventBox*. Then the event signals can be connected on the event box instead of directly to the image.

Any image file with supported type can be loaded, but GTK+ supports also so called stock icons. These stock icons are predefined images which change based on selected theme and can be loaded with special functions. Source code below shows how to load a warning (“gtk-dialog-warning") stock icon.

```
GtkWidget *image1;
image1 = gtk_image_new_from_stock
    ("gtk-dialog-warning", GTK_ICON_SIZE_DIALOG);
gtk_widget_show (image1);
```

GtkLabel is used to display medium amount of texts. Most labels are used to label another widget such as a *GtkButton*, a *GtkMenuItem*, or a *GtkComboBox*.

To make label formatting (changing colors, fonts, etc.) easy, the text inside label can be provided in a simple markup format. Here's how to create a label and mark part of its text with bold font, as shown in Figure 17.

```
label1 = gtk_label_new (NULL);
gtk_label_set_markup (GTK_LABEL (label1),
    "This is a <b>GtkLabel</b>");
```

GtkStatusbar is usually placed along the bottom of an application's main *GtkWindow*. It may provide a regular commentary of the application's status by showing messages when the status changes. It may also have a resize grip (a triangular area in the lower right corner of Figure 17) which can be clicked on to resize the window containing the statusbar.

Statusbars in GTK+ maintain a stack of messages. The message at the top of the each bar's stack is the one that will currently be displayed. Source code below demonstrates how to create a statusbar and add message to it. The result was shown in Figure 17.

```
statusbar1 = gtk_statusbar_new ();
gtk_widget_show (statusbar1);
gtk_statusbar_push (statusbar1, 0,
```

```
    "This is a GtkStatusbar");
```

5.5 Menus

Menus are basic part of modern applications and GTK+ naturally also has its own menu widgets. *GtkMenu* implements a drop down menu consisting of a list of *GtkMenuItem* objects which can be navigated and activated by the user to perform application functions. *GtkMenu* is most commonly dropped down by activating a *GtkMenuItem* in a *GtkMenuBar*, but other composite widgets such as the *GtkNotebook* can pop up a *GtkMenu* as well.

GtkMenuBar is a standard top-menu widget for applications. Figure 18 shows *GtkMenuBar* with several different *GtkMenuItems*.

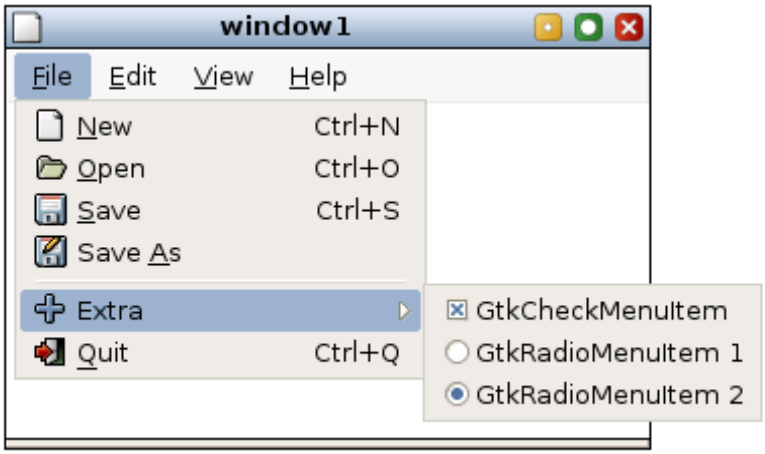


Figure 18. GtkMenuBar with different GtkMenuItems

GtkMenuItem is a base class for all items inside *GtkMenuBar*. *GtkMenuItem* itself only shows text entry's in menu, but it has several sub-widgets which can do more:

- *GtkImageMenuItem* – Menu item which has an icon next to the text label.
- *GtkCheckMenuItem* – Menu item which maintains the state of a boolean value. A check box indicating the state of the boolean value is displayed at the left side of the widget and activating it toggles the value.

- *GtkRadioMenuItem* – Radio menu item is a check menu item that belongs to a group. Only one of the radio menu items of the group cam be selected at the same time.
- *GtkSeparatorMenuItem* – a separator which is used to group items within a menu. It displays a horizontal line with a shadow to make it appear sunken into the interface.

Instead of generating menus manually, they can be also constructed from an XML description since GTK+ 2.4. Using this GtkUIManager makes modifying menu structures simpler and changing them doesn't even require recompilation. More information about this and other menu features are available in /1/.

5.6 Toolbars

The main GTK+ toolbar widget is called *GtkToolbar*. It has two main properties: *orientation* which specifies whether toolbar is horizontal or vertical, and *toolbar-style* which specifies if toolbar items include icons, text, or both. Figure 19 shows a vertical toolbar with both icons and text. Essential parts of source code for creating this are shown later in this chapter.

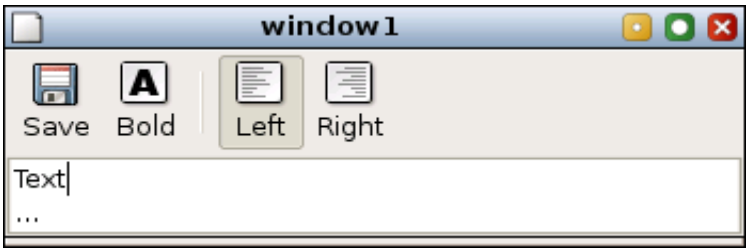


Figure 19. GtkToolbar

GtkToolItems are the special widgets which are attached to *GtkToolbar*. The different toolbar items available in GTK+ 2.6 are:

- *GtkToolButton* – Normal toolbar button which can be clicked. 'Save' button in the example is this kind of button.

- *GtkToggleToolButton* – Toolbar button which can be toggled between two different states. Example of this kind of usage is 'Bold' button in editor, it can be either enabled of disabled.
- *GtkRadioToolButton* – Toolbar button which can be toggled between two different state, with the addition that only one of the buttons in a group can be selected at the same time. Example of this kind on usage is 'Left' and 'Right' alignment, only one of them can be selected at the same time.
- *GtkMenuToolButton* - A *GtkToolItem* that contains a button and a small additional button with an arrow. When clicked, the arrow button pops up a dropdown menu for additional selections.
- *GtkSeparatorToolItem* – Toolbar item which is used as a separator between real items. This can have visible separation line on or it can be just an empty space.
- *GtkToolItem* – The base class of widgets that can be added to *GtkToolbar*. By using this as a parent widget, all different kinds of widgets can be inserted inside *GtkToolbar*.

The source below is the one used to make Figure 19.

```
/* Variables */
GtkWidget *toolbar1;
GtkWidget *toolbutton1;
GtkWidget *toggletoolbutton1;
GtkWidget *separatortoolitem1;
GtkWidget *radiotoolbutton1;
GtkWidget *radiotoolbutton2;
GSList *radiotoolbutton1_group = NULL;

/* Create toolbar */
toolbar1 = gtk_toolbar_new ();
gtk_box_pack_start (GTK_BOX (vbox1),
    toolbar1, FALSE, FALSE, 0);
gtk_toolbar_set_style (GTK_TOOLBAR (toolbar1),
    GTK_TOOLBAR_BOTH);
```

```
/* Create toolbar button */
toolbutton1 = (GtkWidget*)
    gtk_tool_button_new_from_stock ("gtk-save");
gtk_container_add (GTK_CONTAINER (toolbar1),
    toolbutton1);
/* Create toolbar togglebutton */
toggletoolbutton1 = (GtkWidget*)
    gtk_toggle_tool_button_new_from_stock ("gtk-bold");
gtk_container_add (GTK_CONTAINER (toolbar1),
    toggletoolbutton1);
/* Create toolbar separator */
separatortoolitem1 = (GtkWidget*)
    gtk_separator_tool_item_new ();
gtk_container_add (GTK_CONTAINER (toolbar1),
    separatortoolitem1);
/* Create toolbar radiobutton */
radiotoolbutton1 = (GtkWidget*)
    gtk_radio_tool_button_new_from_stock (
    NULL, "gtk-justify-left");
gtk_container_add (GTK_CONTAINER (toolbar1),
    radiotoolbutton1);
gtk_radio_tool_button_set_group (GTK_RADIO_TOOL_BUTTON
    (radiotoolbutton1), radiotoolbutton1_group);
radiotoolbutton1_group =
    gtk_radio_tool_button_get_group (
    GTK_RADIO_TOOL_BUTTON (radiotoolbutton1));
/* Create second radiobutton with same group */
radiotoolbutton2 = (GtkWidget*)
    gtk_radio_tool_button_new_from_stock (NULL,
    "gtk-justify-right");
gtk_container_add (GTK_CONTAINER (toolbar1),
    radiotoolbutton2);
gtk_radio_tool_button_set_group (GTK_RADIO_TOOL_BUTTON
    (radiotoolbutton2), radiotoolbutton1_group);
radiotoolbutton1_group =
    gtk_radio_tool_button_get_group (
    GTK_RADIO_TOOL_BUTTON (radiotoolbutton2));
```

```
/* Show whole toolbar */
gtk_widget_show_all(toolbar1);
```

5.7 Buttons and Toggles

Buttons and toggles are used to launch events and to set different properties. Figure 20 presents the button types available in GTK+.

Figure 20. Different buttons and toggles

GtkButton widget is generally used to attach a function which is called when the button is pressed. It can hold any valid child widget inside, so almost any other standard *GtkWidget*. The most commonly used child's are the *GtkLabel* to show text buttons and *GtkImage* to show buttons with icon.

GtkToggleButton is a *GtkButton* which will remain 'pressed-in' when clicked. Clicking it again will cause the toggle button to return into its default state.

GtkCheckButton is inherited from *GtkToggleButton* with two main differences: It looks like a checkbox instead of a normal button and it has automatically describing widget (usually *GtkLabel*) next to it.

GtkRadioButton is one way of giving the user a choice from many options. A single radio button performs the same basic function as a *GtkCheckButton*, but every radio button is also a member of some group of radio buttons. When one is selected, all other radio buttons in the same group are deselected.

5.8 Data entries

Data entry widgets are an interactive way to get input from the application user. There are four widgets in GTK+ which can be listed to belong in this section: *GtkEntry*, *GtkComboBox*, *GtkComboBoxEntry* and *GtkSpinButton*. Figure 21 shows an example application UI with all these widgets.

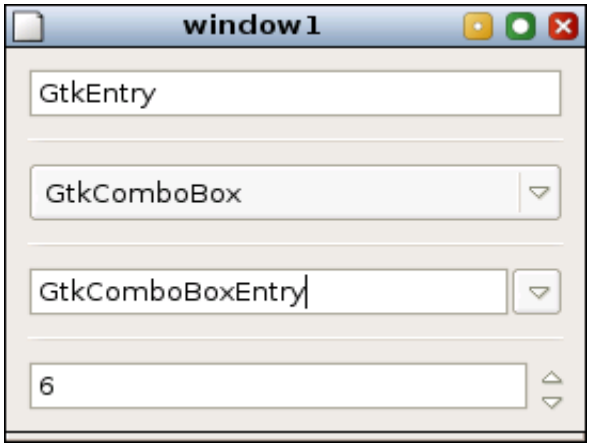


Figure 21. Different data entry widgets

GtkEntry is a single line text entry widget. If the entered text is longer than the allocation of the widget, it will scroll so that the cursor position is visible.

GtkComboBox is a widget that allows the user to choose from a list of valid choices. When activated, a popup menu is displayed which allows the user to make a new choice. The style in which the selected value is displayed and the style of the popup, is determined by the current theme. Unlike its predecessors *GtkCombo* (which is deprecated since GTK+ 2.4) *GtkComboBox* uses the model-view pattern. The list of valid choices is specified in the form of a tree model, and the display of the choices can be adapted to the data in the model by using cell renderer's, same way as in a tree view. The tree model holding the valid choices is not restricted to a flat list, it can be a real tree where popup will reflect the tree structure.

GtkComboBoxEntry is a widget that allows the user to choose from a list of valid choices or enter a different value. It is very similar to a *GtkComboBox*, but it displays the selected value in an entry to allow modifying it. In contrast to a

GtkComboBox, the underlying model of a *GtkComboBoxEntry* must always have a text column, and the entry will show the content of the text column in the selected row.

GtkSpinButton is an ideal way to allow the user to set the numerical value of some attribute. Rather than having to directly type a number into a *GtkEntry, GtkSpinButton* allows the user to click on one of two arrows to increment or decrement the displayed value. A value can also be typed in and checked to ensure it is in a given range.

5.9 Multiline text editor

GTK+ has an extremely powerful framework for multiline text editing. The primary objects involved in the process are *GtkTextBuffer*, which contains the text being edited, and *GtkTextView* which displays it in UI. Each buffer can be displayed by any number of views. Text in a buffer can be marked with *GtkTextTags* to add styles in it. Figure 22 presents these different parts of GTK+ multiline editor.

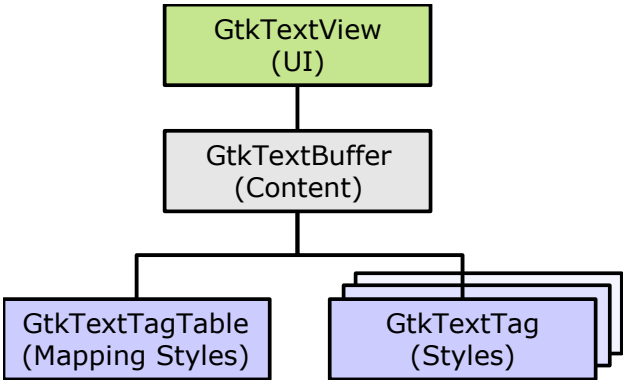


Figure 22. GTK+ multiline text editor parts

GtkTextView is the UI representation of the text view. It is often placed inside *GtkScrolledWindow* to provide scrollbars around the text, so that text can be larger than the available view. Figure 23 shows this kind of combination with styled content.

Figure 23. GtkTextView with mixed content

GtkTextBuffer is the object which contains the model of *GtkTextView*. It uses UTF-8 encoding internally (like most of the GTK+), so its byte counts and character counts differ. *GtkTextBuffer* can contain:

- Text, which thanks to UTF-8 and Pango can be internationalized to contain different character sets.
- *GtkTextTags* with starting and ending iterators (*GtkTextIter*) which hold formatting and information for text buffers. Iterators are places between text characters, so any piece of characters can be tagged.
- Images which are of a type *GdkPixbuf*. They are inserted with gtk_text_buffer_insert_pixbuf function.
- Any sensible GTK+ widgets after making them to *GtkTextChildAnchor*. This possibility to contain any widget makes *GtkTextView* very powerful, when it can be used not only to show content, but also to be a part of interactive user interface.

Source code below demonstrates how to insert image and a *GtkButton* inside *GtkTextBuffer*.

```
/* Variables */
GtkWidget *textview1 = gtk_text_view_new ();
GtkTextBuffer *buffer = gtk_text_view_get_buffer
```

```
    (GTK_TEXT_VIEW (textview1));
GtkTextIter iter;

/* Add image */
GdkPixbuf *pixbuf =
    gdk_pixbuf_new_from_file("lime.png", NULL);
gtk_text_buffer_get_iter_at_offset (buffer, &iter, 0);
gtk_text_buffer_insert_pixbuf(buffer, &iter, pixbuf);
/* Increase iterator with one character */
gtk_text_iter_forward_char(&iter);

/* Add button */
GtkWidget *button1 =
    gtk_button_new_with_mnemonic ("Hit me!");
GtkTextChildAnchor *anchor =
    gtk_text_buffer_create_child_anchor(buffer, &iter);
gtk_text_view_add_child_at_anchor
    ( GTK_TEXT_VIEW(textview1), button1, anchor);
gtk_widget_show (button1);
...
```

A tag is an attribute that can be applied to some range of text. For example, a tag might be called "bold" and make the text inside the tag bold. However, the tag concept is more general than that; tags don't have to affect appearance. They can instead affect the behavior of mouse and key presses, "lock" a range of text so the user can't edit it, or countless other things. A tag is represented by a *GtkTextTag* object. One *GtkTextTag* can be applied to any number of text ranges in any number of buffers. Source code example below shows how to create and apply two different styled tags to text view.

```
/* Prepare widgets */
GtkTextIter start, end;
GtkTextTag *tag;
GtkWidget *textview = gtk_text_view_new ();
GtkTextBuffer *buffer = gtk_text_view_get_buffer
    (GTK_TEXT_VIEW (textview));
```

```
gtk_text_buffer_set_text (buffer,
    ("Make this text bold and blue!"), -1);

/* Select whole text to iterators */
gtk_text_buffer_get_start_iter (buffer, &start);
gtk_text_buffer_get_end_iter (buffer, &end);

/* Create and apply bold tag */
tag = gtk_text_buffer_create_tag (buffer, "bold",
    "weight", 500, NULL);
gtk_text_buffer_apply_tag (buffer,tag, &start, &end);

/* Create and apply blue foregound tag */
tag = gtk_text_buffer_create_tag(buffer, "blue",
    "foreground", "blue", NULL);
gtk_text_buffer_apply_tag (buffer,tag, &start, &end);
```

Each *GtkTextTag* is stored in a *GtkTextTagTable* which defines a set of tags that can be used together. Each buffer has one tag table associated with it; only tags from that tag table can be used with the buffer. A single tag table can be shared between multiple buffers, however.

5.10 Trees and Lists

GTK+ uses MVC (Model-View-Controller) notion for its most complicated widgets, trees and lists. It means that the internal representation of the data (model), the way application shows the data (view), and the ways of manipulating them (controller) are separate objects which communicate with each other /39/.

In GTK+, the tree and list view widgets are combined into *GtkTreeView*. This *GtkTreeView* can then be a flat list like in Figure 24 or tree with subtrees, depending on the model attached to it.

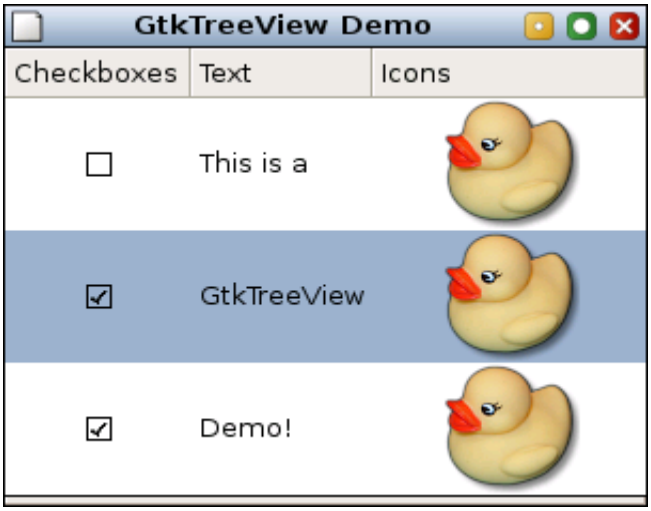


Figure 24. GtkTreeView with mixed content

GtkTreeModel is an interface for building the model. Two simple models which implement it are provided as a default in GTK+: *GtkListStore* and *GtkTreeStore*. *GtkListStore* is used to model list widgets, while the *GtkTreeStore* models trees. It is possible to develop a new type of model, but the existing models should be satisfactory for all but the most specialized situations.

Source code below demonstrates how to create tree widget with different column types. Output of the code it what is visible in Figure 24.

```
/* Structure of treeview */
enum {
  BOOLEAN_COLUMN,
  STRING_COLUMN,
  ICON_COLUMN,
  N_COLUMNS
};

/* Variables */
GtkWidget *window1;
GtkTreeView *treeview1;
GtkListStore *list;
GtkTreeIter iter;
GtkTreeViewColumn *icon_column, *string_column,
    *boolean_column;
```

```
GtkCellRenderer *text_renderer, *toggle_renderer,
    *pixmap_renderer;
GdkPixbuf *pixbuf =
    gdk_pixbuf_new_from_file("duck.png", NULL);

/* Filling list store */
list = gtk_list_store_new(N_COLUMNS, G_TYPE_BOOLEAN,
    G_TYPE_STRING, GDK_TYPE_PIXBUF);
gtk_list_store_append(list, &iter);
gtk_list_store_set(list, &iter,
    BOOLEAN_COLUMN, FALSE,
    STRING_COLUMN, "This is a",
    ICON_COLUMN, pixbuf, -1);
gtk_list_store_append(list, &iter);
gtk_list_store_set(list, &iter,
    BOOLEAN_COLUMN, TRUE,
    STRING_COLUMN, "GtkTreeView",
    ICON_COLUMN, pixbuf, -1);
gtk_list_store_append(list, &iter);
gtk_list_store_set(list, &iter,
    BOOLEAN_COLUMN, TRUE,
    STRING_COLUMN, "Demo!",
    ICON_COLUMN, pixbuf, -1);
treeview1 = GTK_TREE_VIEW (gtk_tree_view_new_with_model(
    GTK_TREE_MODEL(list)));

/* Creating different renderer types */
text_renderer = gtk_cell_renderer_text_new();
toggle_renderer = gtk_cell_renderer_toggle_new();
pixmap_renderer = gtk_cell_renderer_pixbuf_new();
icon_column = gtk_tree_view_column_new_with_attributes
    ("Icons", pixmap_renderer, "pixbuf",
    ICON_COLUMN, NULL);
string_column = gtk_tree_view_column_new_with_attributes
    ("Text", text_renderer, "text",
    STRING_COLUMN, NULL);
boolean_column =
```

```
    gtk_tree_view_column_new_with_attributes
    ("Checkboxes", toggle_renderer, "active",
    BOOLEAN_COLUMN, NULL);

/* Adding colums to treeview */
gtk_tree_view_append_column(treeview1, boolean_column);
gtk_tree_view_append_column(treeview1, string_column);
gtk_tree_view_append_column(treeview1, icon_column);

/* Create window for treeview */
window1 = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window1),
    ("GtkTreeView Demo"));
gtk_window_set_default_size( GTK_WINDOW (window1),
    300, 200);
gtk_container_add(
    GTK_CONTAINER (window1), GTK_WIDGET (treeview1));
gtk_widget_show_all(window1);
```

Cell renderers are used to draw the data in the tree model in a way. There are a number of cell renderers that come with GTK+ 2.x, including the *GtkCellRendererText*, *GtkCellRendererPixbuf* and the *GtkCellRendererToggle*. All these different renderers are used in the above example and shown in Figure 24. Further more, it is relatively easy to write a custom renderer for own usage.

This ends the GTK+ widget chapter. Now all the important widgets have been presented and performance and mobile aspects can be explored next.

6 PERFORMANCE TESTING

This chapter contains GTK+ performance testing and introduces the GtkPerf performance measurement tool created for this thesis to benchmark GTK+ speed in embedded devices.

6.1 Test platforms

6.1.1 Hardware platforms

Hardware used for the testing is listed in Table 4. Testing was done in two different platforms, with Nokia 770 Internet Tablet and with regular X86 compatible PC.

Table 4. Testing platforms hardware

Name	***ARM***	***PC***
Hardware	Nokia 770 Internet Tablet	X86 compatible PC
Processor	Texas Instruments, ARM9-based OMAP 1710, 220 MHz	AMD Athlon 64 3500+ (2200 MHz)
RAM	64 MB	1 GB
Screen resolution	800 x 480 x 16 bpp	1280 x 960 x 16 bpp
Filesystem	128 MB JFFS2 file system partition in flash memory	EXT3 file system on internal hard drive

Originally testing was started with Compaq iPaq H3870 but due to it's limited flash space and screen size, final testing was done with Nokia 770 which is the first Linux-based mobile device from Nokia, shown in Figure 25.

Figure 25. Nokia 770 Internet Tablet

6.1.2 Software platforms

Hardware platforms described in earlier chapter were equipped with software listed in Table 5.

Table 5. Testing platforms software

Name	***ARM***	***PC***
Linux distribution	Nokia Internet Tablet 2005 software edition	Debian/Ubuntu 5.04
Kernel version	2.6.12.3-osso14	2.6.10-5
X-Server version	xserver-kdrive / xserver-xomap_6.6.3-17	XFree86 4.3.0-2.90.55 / xserver-xorg 6.8.2-10
GTK+ versions	libgtk2.0-0 2.6.4-1.osso72 libglib2.0-0 2.6.2-1osso8 libpango1.0-0 1.8.1-1osso3 libatk1.0-0 1.10.1-2	libgtk2.0-0 2.6.4-0ubuntu3 libglib2.0-0 2.6.3-1 libpango-1.0-0 1.8.1-0ubuntu2 libatk1.0-0 1.9.1-0ubuntu1
Other library versions	libfontconfig1 2.3.2-1osso7 libfreetype6 2.1.10-1osso1 hildon-libs0 0.9.52-1 hildon-lgpl0 0.9.55 hildon-base-lib0 0.9.4-1	fontconfig-2.1-9 freetype-2.1.3-6

6.2 GtkPerf testing tool

6.2.1 Introduction

During this thesis there was a need to get measured data of GTK+ performance to see how well GTK+ works in mobile devices with limited processing power. There are plenty of tools available to test CPU speed, but they test just raw processing capabilities. Also tools which test graphical speed are available, but they test X11 drawing or something else non-comparable performance.

It was noticed quite early that there were no testing applications available which would measure the GTK+ performance specifically. This kind of tool should test how fast different widgets are to see how well real GTK+ applications perform in specific hardware and platform setup. Because of this lack of suitable benchmarking software, own tool for this usage was created.

This application, GtkPerf, was then announced with GPL license as freely available for everyone to use. Source code is too big to be added as an appendix in this thesis, so it as well as usage instructions can be found from the GtkPerf website /33/.

GtkPerf was designed to test GTK+ performance on embedded devices, but it can be also helpful to solve other issues:

- How fast is a specific software/hardware platform when compared to others?
- How fast GTK+ is with different themes? Are some widgets notably slower with specific GTK+ theme engines?
- How much new versions of GTK+ improve the performance?
- How different X Servers and configurations affect the GTK+ performance?

GtkPerf was released to publicly available in Gnome development forum and it got a good feedback from users, being an easy to use method to measure GTK+ performance in application level /34/. Later on there were discussion in GTK+ development mailing list about possible performance improvements which supporting Cairo vector graphics library would bring. GtkPerf was considered

there as one tool to measure performance improvements and quite a few GTK+ developers tested it and suggested using it in GTK+ development /35/. Based on the request from GTK+ developers, a new version of GtkPerf was released with improved ability to do automatic test runs.

6.2.2 Operation and usage

GtkPerf operates same way as most of the benchmarking tools: It uses the maximum processor power and measures time used to run the whole test. Whenever one test section is done and CPU starts to idle, new test is started. To be able to achieve this in GTK+ application, GtkPerf takes advantage of GLib event looping features. Figure 26 illustrates how the testing progress jumping to different tests as fast as possible.

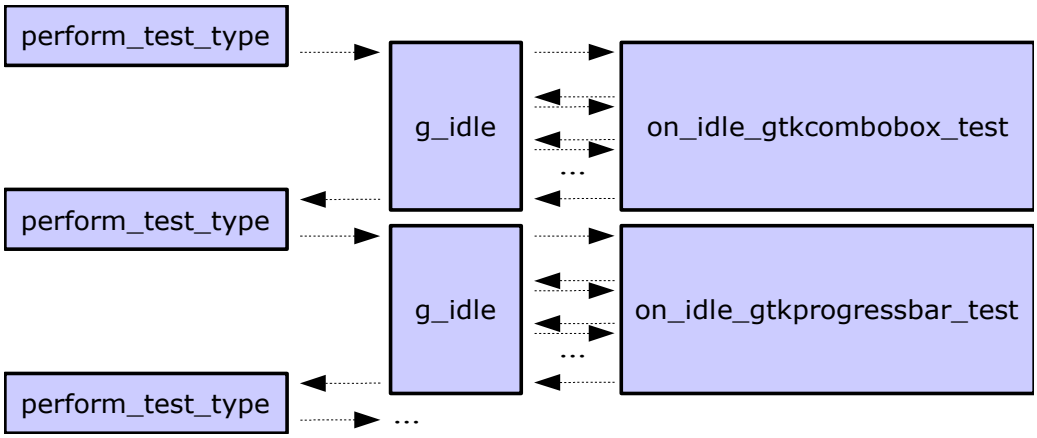


Figure 26. GtkPerf testing progress diagram

Currently GtkPerf contains 14 separate tests which employ 10 different GTK+ widgets as *GtkTextView* and *GtkDrawingArea* have multiple tests. The list of widget tests and description of them is following:

- **GtkEntry** (*on_idle_gtkentry_test*) – Switches between "Test string" and "Longer test string" texts for [count] times.
- **GtkComboBox** (*on_idle_gtkcombobox_test*) – GtkComboBox contains 10 entrys "Selection 1"..."Selection 10". This test opens and closes this GtkComboBox [count] times while selecting the next entry.

- **GtkComboBoxEntry** (*on_idle_gtkcomboboxentry_test*) – GtkComboBoxEntry contains 10 entrys "Selection 1"..."Selection 10". This test opens and closes GtkComboBoxEntry for [count] times while selecting the next entry.
- **GtkSpinButton** (*on_idle_gtkspinbutton_test*) – GtkSpinButton value is increased with one for [count] times. When 1000 is reached, value is changed back to 0.
- **GtkProgressBar** (*on_idle_gtkprogressbar_test*) – GtkProgressBar is increased with 1% for [count] times. Whenever the bar gets full, its value is set to 0.
- **GtkToggleButton** (*on_idle_gtktogglebutton_test*) – GtkToggleButton state is toggled between on/off for [count] times.
- **GtkCheckButton** (*on_idle_gtkcheckbutton_test*) – GtkCheckButton state is toggled between on/off for [count] times.
- **GtkRadioButton** (*on_idle_gtkradiobutton_test*) – Two GtkRadioButtons inside same group are witched to be selected alternately for [count] times.
- **GtkTextView – Add text** (*on_idle_gtktextview_addtext_test*) – Text "Future is Open. " is appended to GtkTextView for [count] times.
- **GtkTextView – Scroll** (*on_idle_gtktextview_scroll_test*) – Scrolls the text added in earlier test GTK_SCROLL_STEP_FORWARD and when the end is reached, GTK_SCROLL_STEP_BACKWARD for [count] times.
- **GtkDrawingArea – Lines** (*on_idle_gtkdrawingarea_lines_test*) – Draws random size and color lines to GtkDrawingArea for 1000*[count] times.
- **GtkDrawingArea – Circles** (*on_idle_gtkdrawingarea_circles_test*) – Draws random size and color circles to GtkDrawingArea for 1000*[count] times.
- **GtkDrawingArea – Text** (*on_idle_gtkdrawingarea_text_test*) – Draws random size and color text “Future is Open” to GtkDrawingArea for 100*[count] times.
- **GtkDrawingArea – Pixbuf** (*on_idle_gtkdrawingarea_pixbufs_test*) – Draws an image (duck.png) to random places in GtkDrawingArea for 10*[count] times.

Main view of GtkPerf is presented in Figure 27. User can select the test count which configures how many times each test is ran. With a slower hardware, smaller count should be used. From the combo box next to count, user can select to perform only a single test case instead of all of them. Results of test run come visible to the text area of main view.

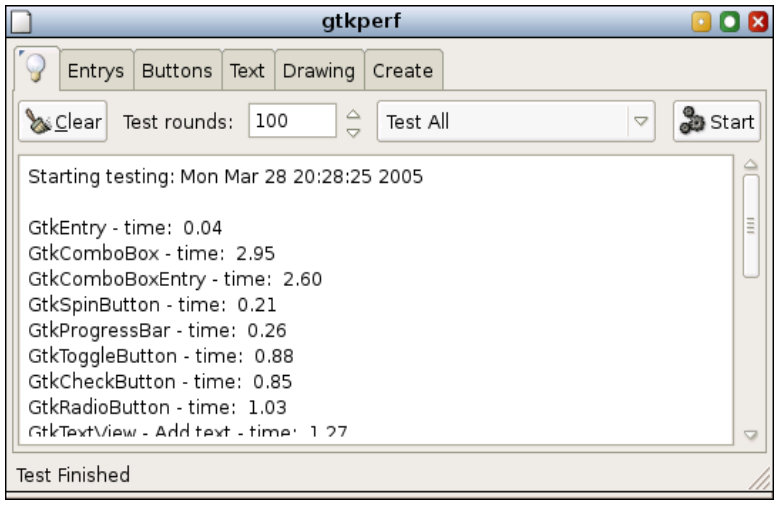


Figure 27. GtkPerf main view

Testing starts with the "Start" button on top right corner. Depending on selected test count and used hardware, running the tests can take several minutes.

Below are two screenshots (Figure 28 and Figure 29) which show GtkPerf running the tests. First one utilizes different entry widgets and the second is drawing images on *GtkDrawingArea*.

Figure 28. GtkPerf entry widget testing

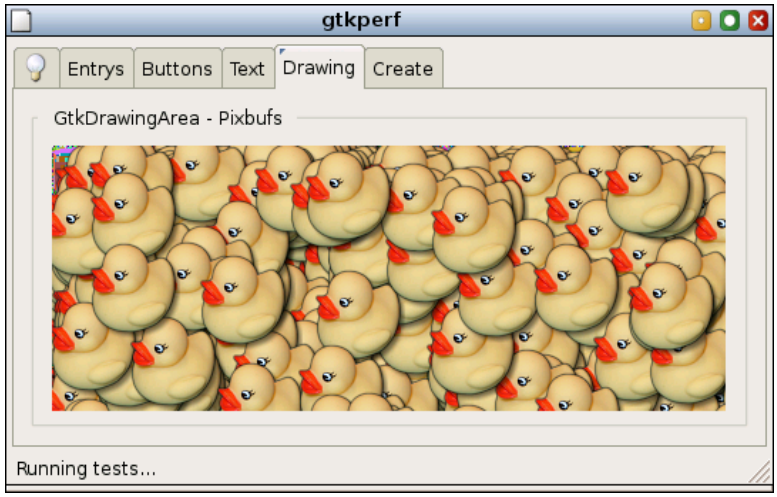


Figure 29. GtkPerf pixbuf testing

6.2.3 GTK+ mobile performance results

GtkPerf was used to measure the GTK+ performance of Nokia 770 Linux device specified in the earlier chapter. These tests were run 3 times with a test count of 200, using the command presented below.

```
# run-standalone.sh /usr/bin/gtkperf -a -c 200 &
```

Tests were run with both Nokia Hildon theme and with GTK+ default theme (Figure 30), to be able to test how much the used theme affects to UI speed. As the Hildon theme is not available currently in other devices, including also default GTK+ theme testing makes the results more easily comparable.

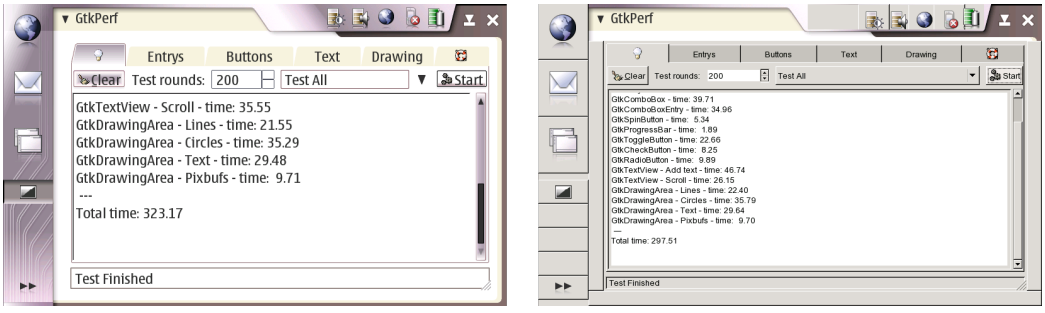


Figure 30. GtkPerf with Hildon theme (left) and with GTK+ default theme

Average running times of this test are presented in Table 6 and a graph with the same data is in Figure 31. The difference column in the table means how much more time the same test takes when Hildon theme is used.

Table 6. GtkPerf results in Nokia 770

Test type	*Hildon theme*	*GTK+ default theme*	*difference*
GtkEntry	2.76s	4.05s	-32.0%
GtkComboBox	44.65s	38.56s	15.8%
GtkComboBoxEntry	42.24s	33.93s	24.5%
GtkSpinButton	7.77s	5.57s	39.4%
GtkProgressBar	2.48s	1.90s	30.1%
GtkToggleButton	10.28s	22.59s	-54.5%
GtkCheckButton	9.25s	8.14s	13.6%
GtkRadioButton	12.41s	9.79s	26.8%
GtkTextView – Add text	52.91s	46.27s	14.4%
GtkTextView – Scroll	36.63s	26.88s	36.3%
GtkDrawingArea – Lines	22.09s	22.66s	-2.5%
GtkDrawingArea – Circles	36.01s	36.82s	-2.2%
GtkDrawingArea – Text	30.01s	30.59s	-1.9%
GtkDrawingArea – Pixbufs	9.77s	9.92s	-1.5%
Total	319.25s	297.68s	7.2%

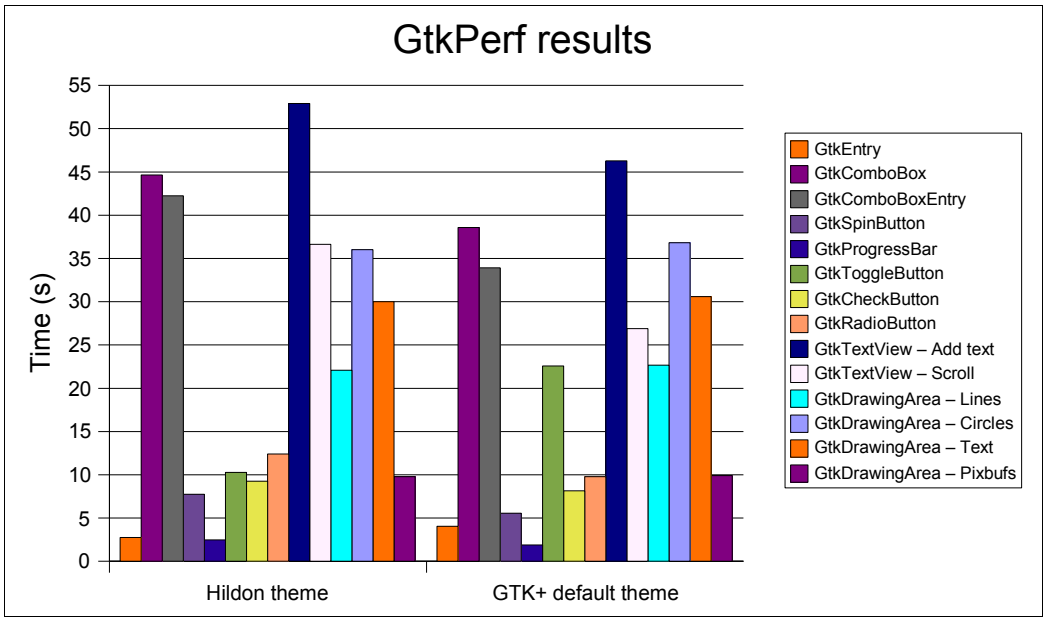


Figure 31. GtkPerf results in Nokia 770

There are several observations which can be made based on these results:

- *GtkComboBox* and *GtkComboBoxEntry* tests are slower with Hildon theme, because default GTK+ theme uses only simple drawing for these combobox widgets instead of images. For the same reasons *GtkSpinButton*, *GtkProgressBar*, *GtkCheckButton* and *GtkRadioButton* tests are faster with simpler default GTK+ theme.
- Differences in *GtkTextView* tests are because of smaller font size of default GTK+ theme. Only conclusion which can be made based on this is that larger fonts are slower to render, but that is quite self-evident anyway.
- All *GtkDrawinArea* tests are equally fast with both themes, which is obvious as themes do not affect to normal GDK drawing speed.
- The reason why *GtkToggleButton* is in fact faster with Hildon theme, is that with that theme toggle buttons there are no graphical changes between different toggle states, only the text of the button minimally changes its location inside the button.

The raw numbers of these results by themselves are not very useful outside this theme comparison. They become more meaningful when compared to results with another hardware or software platforms. When both hardware and software are different, results are again hard to compare. Best usage of GtkPerf is in platform development state, to test how different software modifications and new versions affect the performance. Next section uses this approach by testing the GTK+ performance with different themes.

6.2.4 GTK+ theme performance results

GtkPerf was used to test which GTK+ theme engines and themes are most suitable to embedded devices from the performance point of view. The PC platform defined in chapter 6.1 was used for this test. Total of 17 different GTK+ themes were tested and full results of each widget performance with different themes are available in Appendix B. Table 7 below contains the total times with each theme.

Table 7. GtkPerf results with different themes

Theme	**Total time**
Cleanice	128.73s
Mist	129.02s
GTK+ Default	129.11s
Thinice	129.17s
Simple	129.33s
Redmond95	130.13s
Xfce	131.14s
MilkMint	131.78s
Human	132.39s
Industrial	132.82s
Glider	132.89s
Smooth-Retro	133.46s
Crux	133.79s
Smokey-Blue	161.94s
Gonxical	180.92s
Grand-Canyon	190.56s
eXperience	389.69s

With these results, the conclusion is that *Smokey-Blue, Gonxical, Grand-Canyon* and *eXperience* themes should not be used in embedded devices. When performance and UI responsiveness is important, these themes will give a notably slowdown.

The positive result is that there are also theme engines (*Cleanice, Mist, Thinice, Simple*) which are just as fast as the GTK+ default theme. So nice looking theme can also be fast when it is done right.

6.3 Libglade test

Glade is a free user interface builder for GTK+ and GNOME. The user interfaces designed in Glade are saved as XML, and can be loaded dynamically by applications with the libglade library. Glade can also generate GUI directly to C code, but that is not recommended as the operation only works in one way.

Changes made to GUI source codes will then be overwritten when edited again with Glade /36/.

The second application created for this thesis tests how much the usage of libglade decreases startup performance of GTK+ applications when the user interface needs to be parsed from XML file at runtime on the fly. Libglade also connects automatically the signal handlers in the user interface. Although this helps to separate the UI and functionality from each other and allows Glade to be used with numerous programming languages other than C, it will also grow the application startup time.

Text editor was selected as a suitable application for this test. User interface was built with Glade and then separated to two different applications, one which source code was generated and one which UI was loaded with libglade. So in the end there were two applications which same GUI, shown in Figure 32. This is quite ordinary GUI for an text editor, using *GtkTextView* for text area and toolbar for modifying text styles.

Figure 32. GtkMEdit application built with Glade

Application startup times were then measured three times for both versions, and the average times can be seen in Table 8.

Table 8. Libglade performance test results

	generated code	***libglade***	***difference***
Building GUI	3.08s	3.96s	+28%
GTK+ idle	3.79s	4.65s	+23%

This test shows that building the GUI using libglade adds about 23% to total startup time until the application is in idle, than when generating the C-source with Glade. If customization is considered to be important, this is not too big performance hit. But otherwise libglade should not be used in embedded applications, until it has been optimized some more to embedded aspects. One way to improve its performance would be to convert the UIs in binary mode which could be cached easier than XML files whose parsing is quite time consuming operation.

7 CONCLUSIONS

7.1 GTK+ suitability for mobile devices

Now when all the necessary aspects of mobile operating systems, Linux and GTK+ have been introduced, the essential conclusion part of this thesis begins. This chapter uses the collected information and is dedicated for the GTK+ platform suitability to embedded devices with limited resources and special needs.

Suitability of GTK+ to mobile devices comes down to these issues:

- **Performance:** Evolutionarily GTK+ 1.x was fast, mostly due its simplicity, but when the GTK+ 2.x was introduced the performance got decreased. This was mostly because of much more powerful features, but also partly lack of optimization in several new libraries. GTK+ 2.0 was cleanly more concentrated on new features and now when it matures to GTK+ 2.6 and 2.8, performance bottlenecks are tackled one-by-one.
- **Memory consumption:** Possibly even a bigger obstacle for using GTK+ in mobile devices than performance, is the RAM memory consumption. This has been worked on recently as GTK+ is targeted now also to more limited devices. Especially memory spots which are non-shared between processes and appear in every GTK+ process have been trimmed down. But the problem still exists and GTK+ should not currently be considered to devices with too limited memory (<32 MB).
- **User Interface:** Mobile devices usually include quite a different user interface than the desktop computers which GTK+ is designed to. Thanks to scalability of GTK+ containers and the powerful theming possibilities, these special needs can be mostly fulfilled without changes to GTK+ libraries themselves. Widgets that need to be modified are different selector dialogs which don't work in smaller displays.
- **Platform:** This section is not really about GTK+ itself but the whole platform required for it to be used. This basically includes Linux kernel, X-Server, FreeType and other libraries GTK+ relies on. It is important to

consider their suitability too as it won't matter how good GTK+ itself is if some other platform part makes it unusable in embedded devices.

7.2 Results of the thesis

The result of this thesis is that GTK+ and Linux are scalable enough to be used also in mobile devices. Many of the problems in GTK+ have been spotted and fixed for GTK+ 2.6 or 2.8 versions and it is maturing to be a real competitor in the embedded area.

One evidence of this suitability is the maemo platform created by Nokia, which uses modified GTK+ toolkit for the user interface layer. Maemo platform is meant for more higher-end handhelds and the first device using it is Nokia 770 Internet Tablet. The biggest reasons why Nokia decided to use GTK+ in maemo are its wide support in open source community, large amount of available applications and a suitable license for both open and commercial development.

7.3 Future work

There are several things which should be investigated to evaluate further what kind of future GTK+ and Linux will have in embedded world:

- Newer GTK+ versions include changes and additions, whose suitability for embedded usage must be tested. For example GTK+ 2.8 already has a dependency to new rendering library called Cairo, which is a 2D drawing library supporting graphic acceleration. It can render the graphics to both display and printer and includes multiple output formats. Testing the Cairo and optimizing it to embedded devices with most suitable rendering backends and HW acceleration should be done. This will help to improve GTK+ performance and make it more graphically appealing for application developers and users.
- At some point GTK+ may need separation for current desktop version and to more limited embedded version. This would mean that most common widgets would belong to both versions, but embedded GTK+ would not

contain everything and it would include special versions of some widgets. This is something which has already been done in maemo platform with Hildon widgets /41/. This separation should be done more cleanly in GTK+ with different configurations, so that embedded version would not be a separate fork.

- GtkPerf testing tool developed for this thesis has received some interest and should be further improved. More widget tests could be made, maybe some for testing Cairo drawing performance introduced earlier. GTK+ developers have also requested a possibility to give more testing settings as command line parameters, to be able to run specific automatic tests. All these improvements should be doable and GtkPerf would then further help the GTK+ development.

The future of GTK+ in mobile devices remains unclear, and it also depends much on how Linux succeeds in that area. But they both definitely have their own strong points, and big mobile device companies are the ones which will decide their longer term success.

REFERENCES

/1/ GTK+ API Reference Manual [Internet]. Location: http://developer.gnome.org/doc/API/2.0/gtk/ [Referenced 5.2.2005]

/2/ Warkus, Matthias. The Official GNOME 2 Developer's Guide. No Starch Press, Inc. First edition, April 2004. ISBN 1-59327-030-5.

/3/ maemo development platform [Internet]. Location: http://maemo.org/ [Referenced 27.6.2005]

/4/ Gartner. Market Share: Mobile Terminals, Worldwide, 1Q05. Available from: http://linuxdevices.com/news/NS8804000399.html [Referenced 1.12.2005]

/5/ IDC. Worldwide Mobile Phone 2005-2009 Forecast and Analysis. Available from: http://www.linuxdevices.com/news/NS4058662049.html [Referenced 1.12.2005]

/6/ Symbian, Ltd. [Internet]. Location: http://www.symbian.com [Referenced 1.8.2004]

/7/ Microsoft Ltd. Windows Mobile edition [Internet]. Location: http://www.microsoft.com/windowsmobile [Referenced 5.10.2004]

/8/ PalmOne, inc. [Internet]. Location: http://www.palmone.com/ [Referenced 21.1.2005]

/9/ PalmSource. PalmSource to Extend Leadership in Phone Software with Acquisition of China MobileSoft [Internet]. Location: http://www.palmsource.com/press/2004/120804_cms.html [Referenced 21.1.2005]

/10/ Kernel.org - Linux kernel archives [Internet]. Location: http://www.kernel.org [Referenced 19.2.2005]

/11/ Joseph Pranevich. The Wonderful World of Linux 2.6 [Internet]. Location: http://www.kniggit.net/wwol26.html [Referenced 8.7.2005]

/12/ GNU General Public license. Available from: http://www.gnu.org/licenses/gpl.html [Referenced 28.3.2005]

/13/ LinuxDevices.com – Snapshot of the embedded Linux market, March 2004 [Internet]. Location: http://linuxdevices.com/articles/AT8693703925.html [Referenced 19.2.2005]

/14/ MontaVista Software, inc. [Internet]. Location: http://www.mvista.com [Referenced 28.2.2005]

/15/ LynuxWorks, inc. [Internet]. Location: http://www.lynuxworks.com/ [Referenced 28.2.2005]

/16/ Metrowerks, Ltd. Linux solutions [Internet]. Location: http://www.metrowerks.com/MW/Develop/Embedded/Linux/ [Referenced 22.1.2005]

/17/ GPE Palmtop Environment [Internet]. Location: http://gpe.handhelds.org [Referenced 19.2.2005]

/18/ Opie - Open Palmtop Integrated Environment [Internet]. Location: http://opie.handhelds.org [Referenced 19.2.2005]

/19/ OpenEmbedded platform [Internet]. Location: http://www.openembedded.org [Referenced 24.1.2005]

/20/ endebian, Embedded Debian project [Internet]. Location: http://www.emdebian.org/ [Referenced 22.1.2005]

/21/ Glib Object system tutorial [Internet]. Location: http://le-hacker.org/papers/gobject/ [Referenced 13.3.2005]

/22/ Eero Tamminen, Veli Mankinen, Lauri Leukkunen, Erik Andersen. Cross-compilation and cross-configuration. Available from: http://scratchbox.org/pdf/cross-conf.pdf [Referenced 18.4.2005]

/23/ Mankinen, Veli. An introduction to the Scratchbox cross-development tool. Available from: http://www.movial.fi/client-data/file/20040330_Movial_Scratchbox_white_paper.pdf [Referenced 17.7.2004]

/24/ Scratchbox SDK [Internet]. Location: http://www.scratchbox.org [Referenced 7.7.2004].

/25/ Amundson Shawn T., Deloget Emmanuel, Gale Tony. GTK+ Frequently Asked Questions, Available from: http://www.gtk.org/faq/ [Referenced 2.5.2004]

/26/ The GIMP Toolkit (GTK+) official website [Internet]. Location: http://gtk.org/ [Referenced 12.5.2004]

/27/ FreeType font rendering engine [Internet]. Location: http://freetype.org [Referenced 13.3.2005]

/28/ The GTK+ GLib tutorial. Available from: http://www.gtk.org/tutorial/ch-glib.html [Referenced 2.5.2004]

/29/ Pango text layout engine [Internet]. Location: http://www.pango.org [Referenced 27.3.2005]

/30/ ATK accessibility toolkit API documentation [Internet]. Location: http://developer.gnome.org/doc/API/2.0/atk/ [Referenced 27.3.2005]

/31/ Gnome theming tutorial and HOWTO [Internet]. Location: http://ajgenius.us/Gnome/Themes/Tutorials/ [Referenced 30.10.2004]

/32/ GTK+ Resource files, API documentation [Internet]. Location: http://developer.gnome.org/doc/API/2.4/gtk/gtk-Resource-Files.html [Referenced 31.10.2004]

/33/ GtkPerf performance testing tool [Internet]. Location: http://gtkperf.sourceforge.net [Referenced 16.6.2005]

/34/ GnomeDesktop.org -GtkPerf testing tool released [Internet]. Location: http://gnomedesktop.org/node/2277 [Referenced 16.6.2005]

/35/ GTK+ development mailing list, subject: “gtk performance testing“ [Internet]. Location: http://mail.gnome.org/archives/gtk-devel-list/2005-June/msg00069.html [Referenced 22.8.2005]

/36/ Glade GUI builder [Internet]. Location: http://glade.gnome.org [Referenced 28.08.2005]

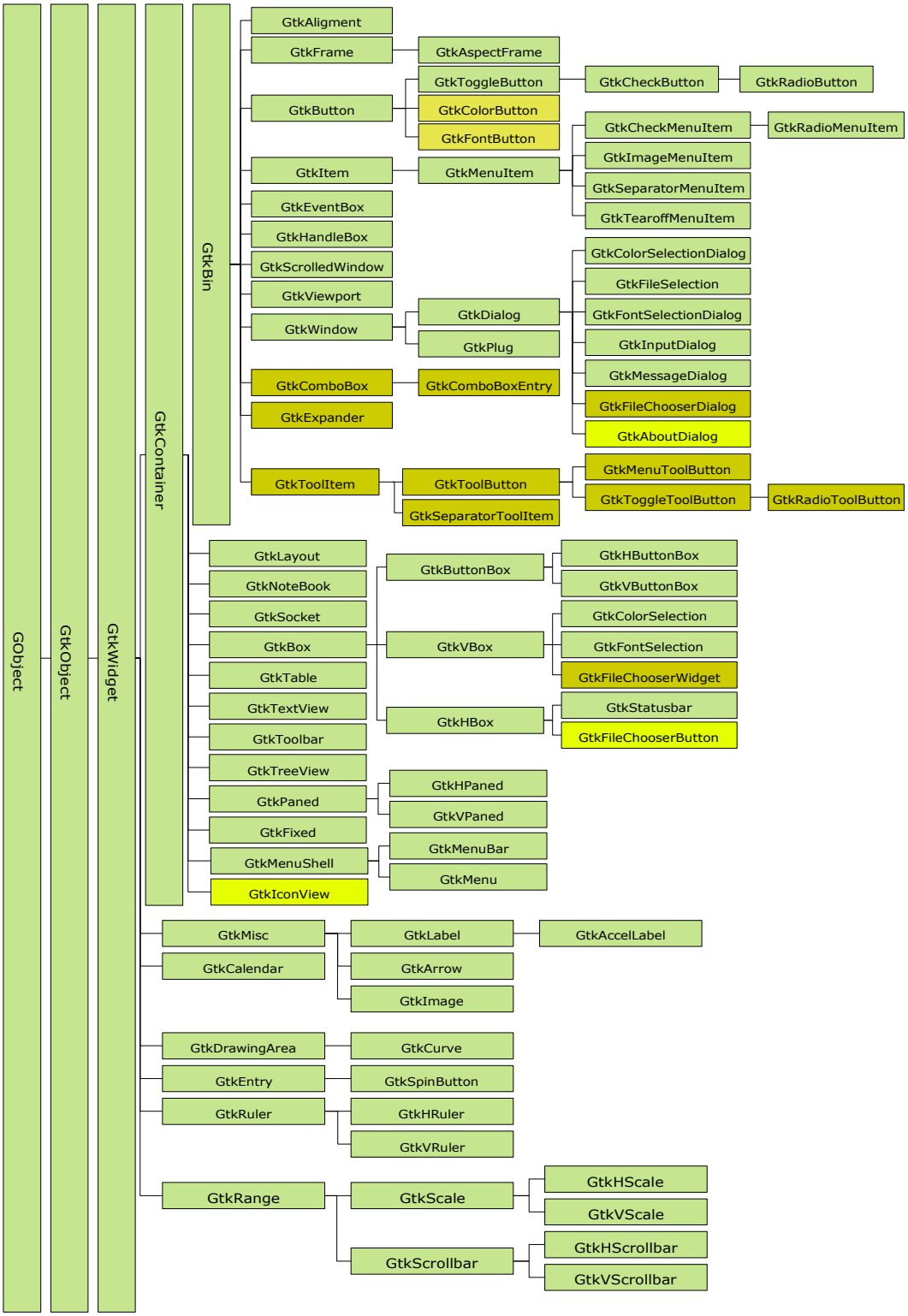
/37/ Yaghmour, Karim. Building Embedded Linux Systems. O'Reilly & Associates. First edition, April 2003. ISBN 0-596-00222-X.

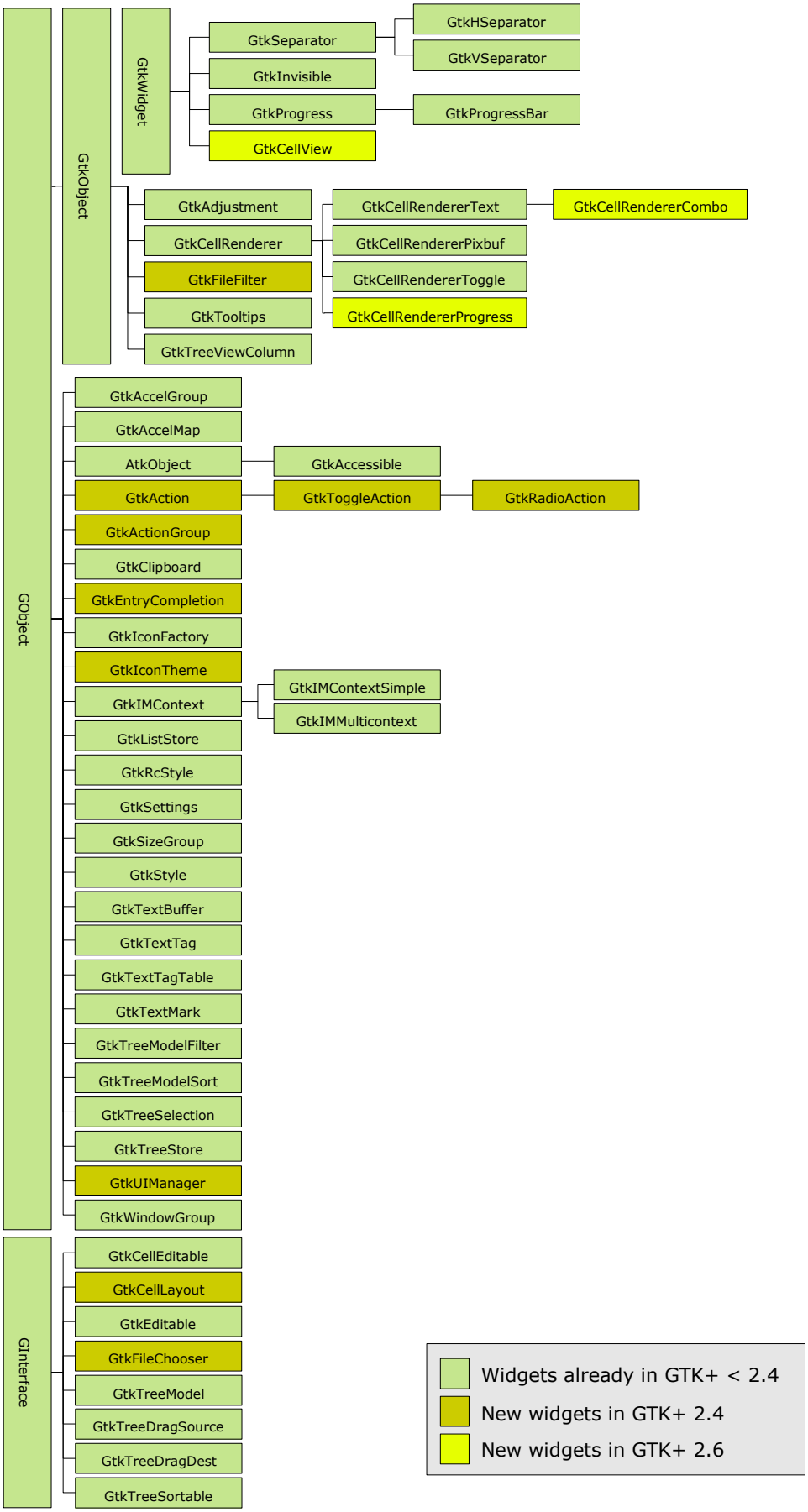
/38/ Wind River Systems, Inc. [Internet]. Location: http://www.windriver.com [Referenced 12.10.2005]

/39/ Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. Pattern-oriented software architecture: A system of patterns. John Wiley & Sons, 1996. ISBN 0-471-95869-7.

/40/ Roger S. Pressman. Software Engineering. McGraw-Hill Publishing Company. Fifth edition, 2000. ISBN 0-07-709677-0.

APPENDIX A. GTK+ WIDGET HIERARCHY




GObject
GtkObject
GtkWidget
GtkSeparator
GtkHSeparator
GtkVSeparator
GtkInvisible
GtkProgress
GtkProgressBar
GtkCellView
GtkAdjustment
GtkCellRenderer
GtkCellRendererText
GtkCellRendererCombo
GtkCellRendererPixbuf
GtkCellRendererToggle
GtkCellRendererProgress
GtkFileFilter
GtkTooltips
GtkTreeViewColumn
GtkAccelGroup
GtkAccelMap
AtkObject
GtkAccessible
GtkAction
GtkToggleAction
GtkRadioAction
GtkActionGroup
GtkClipboard
GtkEntryCompletion
GtkIconFactory
GtkIconTheme
GtkIMContext
GtkIMContextSimple
GtkIMMulticontext
GtkListStore
GtkRcStyle
GtkSettings
GtkSizeGroup
GtkStyle
GtkTextBuffer
GtkTextTag
GtkTextTagTable
GtkTextMark
GtkTreeModelFilter
GtkTreeModelSort
GtkTreeSelection
GtkTreeStore
GtkUIManager
GtkWindowGroup
GInterface
GtkCellEditable
GtkCellLayout
GtkEditable
GtkFileChooser
GtkTreeModel
GtkTreeDragSource
GtkTreeDragDest
GtkTreeSortable
Widgets already in GTK+ < 2.4
New widgets in GTK+ 2.4
New widgets in GTK+ 2.6

APPENDIX B. GTKPERF THEME RESULTS

```
Test rounds: 1000

Gonxical
----------
Starting testing: Sun Feb  6 17:34:17 2005
GtkEntry - time:  0.48
GtkComboBox - time: 29.34
GtkComboBoxEntry - time: 25.90
GtkSpinButton - time:  1.97
GtkProgressBar - time:  2.50
GtkToggleButton - time:  8.66
GtkCheckButton - time:  8.53
GtkRadioButton - time: 10.47
GtkTextView - Add text - time: 53.06
GtkTextView - Scroll - time: 19.46
GtkDrawingArea - Lines - time:  2.15
GtkDrawingArea - Circles - time:  5.16
GtkDrawingArea - Text - time:  6.52
GtkDrawingArea - Pixbufs - time:  6.71
 ---
Total time: 180.92

Human
----------
Starting testing: Sun Feb  6 17:44:12 2005
GtkEntry - time:  0.50
GtkComboBox - time:  9.09
GtkComboBoxEntry - time:  9.77
GtkSpinButton - time:  0.76
GtkProgressBar - time:  0.58
GtkToggleButton - time:  2.66
GtkCheckButton - time:  2.35
GtkRadioButton - time:  2.81
GtkTextView - Add text - time: 53.47
GtkTextView - Scroll - time: 24.01
GtkDrawingArea - Lines - time:  2.72
GtkDrawingArea - Circles - time:  8.05
GtkDrawingArea - Text - time:  8.78
GtkDrawingArea - Pixbufs - time:  6.82
 ---
Total time: 132.39

Cleanice
----------
Starting testing: Sun Feb  6 17:47:31 2005
GtkEntry - time:  0.48
GtkComboBox - time:  8.37
GtkComboBoxEntry - time:  8.86
GtkSpinButton - time:  0.73
GtkProgressBar - time:  0.59
GtkToggleButton - time:  2.19
GtkCheckButton - time:  2.00
GtkRadioButton - time:  2.48
GtkTextView - Add text - time: 53.07
GtkTextView - Scroll - time: 23.55
GtkDrawingArea - Lines - time:  2.74
GtkDrawingArea - Circles - time:  8.07
GtkDrawingArea - Text - time:  8.81
GtkDrawingArea - Pixbufs - time:  6.79
 ---
Total time: 128.73

Crux
----------
Starting testing: Sun Feb  6 17:50:25 2005
GtkEntry - time:  0.47
GtkComboBox - time:  9.14
GtkComboBoxEntry - time:  9.27
GtkSpinButton - time:  0.98
```

```
GtkProgressBar - time:  1.53
GtkToggleButton - time:  2.66
GtkCheckButton - time:  2.45
GtkRadioButton - time:  2.89
GtkTextView - Add text - time: 53.29
GtkTextView - Scroll - time: 24.75
GtkDrawingArea - Lines - time:  2.72
GtkDrawingArea - Circles - time:  8.06
GtkDrawingArea - Text - time:  8.77
GtkDrawingArea - Pixbufs - time:  6.81
 ---
Total time: 133.79

Default
----------
Starting testing: Sun Feb  6 17:53:25 2005
GtkEntry - time:  0.51
GtkComboBox - time:  8.38
GtkComboBoxEntry - time:  8.91
GtkSpinButton - time:  0.76
GtkProgressBar - time:  0.60
GtkToggleButton - time:  2.23
GtkCheckButton - time:  2.07
GtkRadioButton - time:  2.53
GtkTextView - Add text - time: 53.14
GtkTextView - Scroll - time: 23.54
GtkDrawingArea - Lines - time:  2.73
GtkDrawingArea - Circles - time:  8.00
GtkDrawingArea - Text - time:  8.85
GtkDrawingArea - Pixbufs - time:  6.85
 ---
Total time: 129.11

eXperience
----------
Starting testing: Sun Feb  6 17:56:09 2005
GtkEntry - time:  0.49
GtkComboBox - time: 75.68
GtkComboBoxEntry - time: 73.16
GtkSpinButton - time: 19.49
GtkProgressBar - time:  8.11
GtkToggleButton - time: 31.09
GtkCheckButton - time: 26.89
GtkRadioButton - time: 30.54
GtkTextView - Add text - time: 61.67
GtkTextView - Scroll - time: 41.66
GtkDrawingArea - Lines - time:  2.25
GtkDrawingArea - Circles - time:  5.28
GtkDrawingArea - Text - time:  6.51
GtkDrawingArea - Pixbufs - time:  6.87
 ---
Total time: 389.69

Glider
----------
Starting testing: Sun Feb  6 18:03:12 2005
GtkEntry - time:  0.51
GtkComboBox - time:  9.34
GtkComboBoxEntry - time:  9.81
GtkSpinButton - time:  1.09
GtkProgressBar - time:  0.68
GtkToggleButton - time:  2.39
GtkCheckButton - time:  2.20
GtkRadioButton - time:  2.68
GtkTextView - Add text - time: 53.50
GtkTextView - Scroll - time: 24.39
GtkDrawingArea - Lines - time:  2.72
GtkDrawingArea - Circles - time:  8.01
GtkDrawingArea - Text - time:  8.79
GtkDrawingArea - Pixbufs - time:  6.79
 ---
Total time: 132.89
```

```
Grand-Canyon
----------
Starting testing: Sun Feb  6 18:05:58 2005
GtkEntry - time:  0.49
GtkComboBox - time: 22.73
GtkComboBoxEntry - time: 21.50
GtkSpinButton - time:  1.88
GtkProgressBar - time:  3.04
GtkToggleButton - time:  9.23
GtkCheckButton - time:  9.22
GtkRadioButton - time: 12.18
GtkTextView - Add text - time: 57.52
GtkTextView - Scroll - time: 26.36
GtkDrawingArea - Lines - time:  2.75
GtkDrawingArea - Circles - time:  8.09
GtkDrawingArea - Text - time:  8.78
GtkDrawingArea - Pixbufs - time:  6.77
 ---
Total time: 190.56

Industrial
----------
Starting testing: Sun Feb  6 18:09:37 2005
GtkEntry - time:  0.51
GtkComboBox - time:  9.22
GtkComboBoxEntry - time:  9.83
GtkSpinButton - time:  0.74
GtkProgressBar - time:  0.58
GtkToggleButton - time:  2.67
GtkCheckButton - time:  2.36
GtkRadioButton - time:  2.83
GtkTextView - Add text - time: 53.79
GtkTextView - Scroll - time: 24.06
GtkDrawingArea - Lines - time:  2.72
GtkDrawingArea - Circles - time:  7.99
GtkDrawingArea - Text - time:  8.78
GtkDrawingArea - Pixbufs - time:  6.74
 ---
Total time: 132.82

MilkMint
----------
Starting testing: Sun Feb  6 18:12:59 2005
GtkEntry - time:  0.51
GtkComboBox - time:  9.29
GtkComboBoxEntry - time:  9.47
GtkSpinButton - time:  1.01
GtkProgressBar - time:  0.67
GtkToggleButton - time:  2.35
GtkCheckButton - time:  2.07
GtkRadioButton - time:  2.56
GtkTextView - Add text - time: 53.47
GtkTextView - Scroll - time: 24.04
GtkDrawingArea - Lines - time:  2.76
GtkDrawingArea - Circles - time:  8.06
GtkDrawingArea - Text - time:  8.73
GtkDrawingArea - Pixbufs - time:  6.77
 ---
Total time: 131.78

Mist
----------
Starting testing: Sun Feb  6 18:15:42 2005
GtkEntry - time:  0.51
GtkComboBox - time:  8.22
GtkComboBoxEntry - time:  8.63
GtkSpinButton - time:  0.69
GtkProgressBar - time:  0.56
GtkToggleButton - time:  2.02
GtkCheckButton - time:  1.87
GtkRadioButton - time:  2.30
GtkTextView - Add text - time: 53.82
GtkTextView - Scroll - time: 23.89
```

```
GtkDrawingArea - Lines - time:  2.74
GtkDrawingArea - Circles - time:  8.11
GtkDrawingArea - Text - time:  8.82
GtkDrawingArea - Pixbufs - time:  6.83
 ---
Total time: 129.02

Redmond95
----------
Starting testing: Sun Feb  6 18:18:41 2005
GtkEntry - time:  0.48
GtkComboBox - time:  8.52
GtkComboBoxEntry - time:  9.01
GtkSpinButton - time:  0.75
GtkProgressBar - time:  0.60
GtkToggleButton - time:  2.21
GtkCheckButton - time:  2.07
GtkRadioButton - time:  2.55
GtkTextView - Add text - time: 53.38
GtkTextView - Scroll - time: 24.27
GtkDrawingArea - Lines - time:  2.76
GtkDrawingArea - Circles - time:  8.00
GtkDrawingArea - Text - time:  8.77
GtkDrawingArea - Pixbufs - time:  6.74
 ---
Total time: 130.13

Simple
----------
Starting testing: Sun Feb  6 18:21:48 2005
GtkEntry - time:  0.49
GtkComboBox - time:  8.42
GtkComboBoxEntry - time:  8.86
GtkSpinButton - time:  0.74
GtkProgressBar - time:  0.59
GtkToggleButton - time:  2.13
GtkCheckButton - time:  1.99
GtkRadioButton - time:  2.47
GtkTextView - Add text - time: 53.58
GtkTextView - Scroll - time: 23.72
GtkDrawingArea - Lines - time:  2.68
GtkDrawingArea - Circles - time:  8.01
GtkDrawingArea - Text - time:  8.80
GtkDrawingArea - Pixbufs - time:  6.85
 ---
Total time: 129.33

Smokey-Blue
----------
Starting testing: Sun Feb  6 18:24:29 2005
GtkEntry - time:  0.49
GtkComboBox - time: 20.60
GtkComboBoxEntry - time: 13.50
GtkSpinButton - time:  0.89
GtkProgressBar - time:  0.62
GtkToggleButton - time:  8.25
GtkCheckButton - time:  5.24
GtkRadioButton - time:  5.94
GtkTextView - Add text - time: 54.93
GtkTextView - Scroll - time: 24.94
GtkDrawingArea - Lines - time:  2.83
GtkDrawingArea - Circles - time:  8.11
GtkDrawingArea - Text - time:  8.80
GtkDrawingArea - Pixbufs - time:  6.81
 ---
Total time: 161.94

Smooth-Retro
----------
Starting testing: Sun Feb  6 18:27:47 2005
GtkEntry - time:  0.48
GtkComboBox - time:  9.50
GtkComboBoxEntry - time:  9.84
```

```
GtkSpinButton - time:  1.14
GtkProgressBar - time:  0.67
GtkToggleButton - time:  2.44
GtkCheckButton - time:  2.16
GtkRadioButton - time:  2.67
GtkTextView - Add text - time: 53.58
GtkTextView - Scroll - time: 24.58
GtkDrawingArea - Lines - time:  2.70
GtkDrawingArea - Circles - time:  8.08
GtkDrawingArea - Text - time:  8.79
GtkDrawingArea - Pixbufs - time:  6.82
 ---
Total time: 133.46

Thinice
----------
Starting testing: Sun Feb  6 18:30:43 2005
GtkEntry - time:  0.51
GtkComboBox - time:  8.46
GtkComboBoxEntry - time:  8.92
GtkSpinButton - time:  0.77
GtkProgressBar - time:  0.59
GtkToggleButton - time:  2.23
GtkCheckButton - time:  1.95
GtkRadioButton - time:  2.41
GtkTextView - Add text - time: 53.45
GtkTextView - Scroll - time: 23.55
GtkDrawingArea - Lines - time:  2.72
GtkDrawingArea - Circles - time:  8.01
GtkDrawingArea - Text - time:  8.82
GtkDrawingArea - Pixbufs - time:  6.78
 ---
Total time: 129.17

Xfce
----------
Starting testing: Sun Feb  6 18:33:25 2005
GtkEntry - time:  0.48
GtkComboBox - time:  8.84
GtkComboBoxEntry - time:  9.43
GtkSpinButton - time:  0.85
GtkProgressBar - time:  0.69
GtkToggleButton - time:  2.39
GtkCheckButton - time:  2.07
GtkRadioButton - time:  2.70
GtkTextView - Add text - time: 53.44
GtkTextView - Scroll - time: 23.91
GtkDrawingArea - Lines - time:  2.75
GtkDrawingArea - Circles - time:  8.01
GtkDrawingArea - Text - time:  8.80
GtkDrawingArea - Pixbufs - time:  6.77
 ---
Total time: 131.14
```