Gtk v2 Evaluation Report (final version)

Eero Tamminen

2002-10-14

Abstract

Description of the Gtk toolkit version 2 and summary of it's features compared to embedded platform requirements such as size, internationalization, theming and other features.

Contents

[1] http://www.pango.org/
[2] http://www.gnome.org/

List of Figures

Part I
Gtk toolkit

1 Summary

Gnome / Gtk pair is one of the two most popular desktop environments / widget sets used on Linux (KDE / Qt is another). It's in process of replacing the proprietary user interfaces (e.g. Motif / CDE) also on commercial Unix implementations.

Gtk v2 is latest version of Gtk featuring much enhanced internationalization and accessibility framework.

1.1 Advantages

- Modular.
- Has lots of widgets.
- Nice user interface builder.
- "Full" internationalization with Unicode and bidirectional text support.
- As it's other of the two main Open Source widget sets and traditional Unix companies are moving to use it too, it's development and maturity should be guaranteed.
- Glib offers some GUI independent system framework facilities.

1.2 Disadvantages

- Very large size.
- No embedded version or build options for smaller configuration are ready.
- C based object orientation, which is more error prone than a compiler supported one.
- Huge API, learning it will take a lot of time.
- Configuring Gtk components for cross compilation is problematic.

1.3 Conclusions

For embedded use on the low and middle end devices the library would need to be seriously pruned, but it's not yet sure whether it can be made small enough as there are currently no building options for this. Widgets do much more than is required of PDA widgets and several of them should be rewritten for smaller screen sizes.

If Unicode level 3 support is needed, only real alternatives in Open Source are Qt v3 toolkit and Pango component in Gtk v2.

1.4 Missing from evaluation

Some things are missing from this evaluation:

- Locale handling and Unicode input applications for Gtk were not evaluated.
- PDA specific widget navigation issues were not evaluated.
- Available Gtk applications and their adaptability for PDA devices was not evaluated.
- For the previous Gtk version and cross compilation notes, see the older Gtk evaluation from 2001.

2 Components

2.1 Gtk components

Gtk toolkit[3] is composed of several components which are *separate* projects, but released together because of the version interdependencies.

2.1.1 Glib

Glib is a generic utility component which is also used by other projects outside Gtk. Version 2 provides a lot more functionality than the earlier version did:

Application support The main event loop, threads abstraction, thread pooling and queues, dynamic loading of modules, memory allocation, portable IO channels, timers, error reporting, debugging and logging functions.

Utilities Unicode manipulation, character set conversations and string functions, date and time functions, random numbers, hook (callback) manipulation, string completion, lexical scanner, simple XML parser, pattern matching, timers, process spawning and Windows compatibility functions.

Data structures Lists, queues, hashes, stacks, dynamic strings, string chunks, arrays, binary and N-ary trees, quarks, keyed lists, relations, tuples, datasets, memory allocators and caches.

Portability Portable type definitions, type conversions and portability macros that are used everywhere in Glib and Gtk.

Objects Signals (an object notification mechanism), run time type identification and management, type loading and Generic values (a polymorphic type).

Building Glib package produces *glib*, *gobject*, *gmodule* and *gthread* libraries.

[3] http://www.gtk.org/

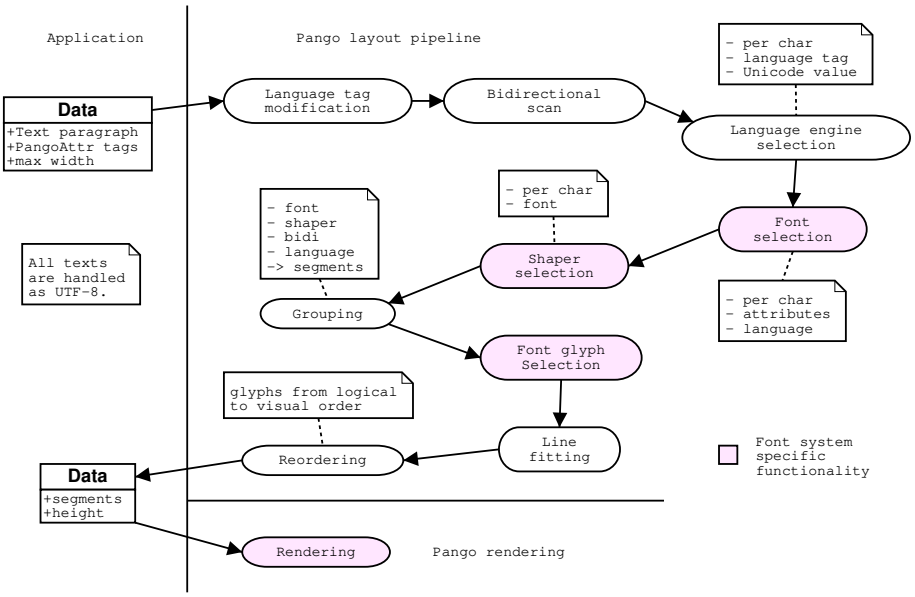


Figure 1: Pango text rendering pipeline

2.1.2 Atk

Atk library provides a set of interfaces for accessibility. There's a different interface for each different type of a widget (valuator, text, image, document etc). By supporting the Atk interfaces, an application or toolkit can be used such as tools such as screen readers, magnifiers, and alternative input devices.

Atk library provides also utilities for streaming object states, type registration and object relation handling.

2.1.3 Pango[4]

Pango is a Unicode text layout and rendering library with bidirectional text routines which is used by all Gtk text output routines. It's very modular, there's a module structure for both layout engines and input methods. Currently Pango has support[5] for X11, WIN32, FreeType and OpenType font systems and includes support for basic (western), arabic, hangul, hebrew, indic, tamil and thai text layout. Pango supports multiple languages and styles (colors, font sizes and types) for single text.

Building Pango package will produce the core pango library, font rendering system specific pango library and runtime loaded language shaper libraries for given rendering system.

[4] http://www.pango.org/

[5] Maturity of the Pango support for different font systems and of different language text layout engines was not evaluated.

2.1.4 GdkPixbuf

GdkPixbuf is an image handling library for Gdk. It provides image loading and saving with dynamically loaded image type handlers, image rendering, manipulation and animation functionality. GdkPixbuf reference counts images and has a hook for letting application optionally to cache the images.

GdkPixbuf is included into the Gtk package and it will produce the GdkPixbuf library and some runtime loaded image format (jpeg, png etc) handler libraries.

2.1.5 Gdk

Gdk is a graphics abstraction library and API for Gtk. It provides graphics contexts, drawing primitives, window, image, font and color manipulation, events, selections, drag and drop, thread and IO channel support on top of Glib. Gdk can work on top of X11, linux frame buffer and WIN32 graphics APIs. There's also a DirectFB version of Gdk, see `http://www.directfb.org/gtk.xml`.

Gdk library is included into Gtk package and like GdkPixbuf the produced library will be specific for the underlying graphics API.

2.1.6 Gtk

Gtk component contains the the actual GUI widgets and framework. Using the underlying Glib, Pango and Gdk APIs, Gtk can work on top of different graphics APIs. Gtk backend alternatives and dependencies are shown in the figure 2.

Widgets are described in section 3.3 and library sizes are given in section 4.2.

2.2 Gnome[6]

Gnome[7] is a desktop environment based on the Gtk widget set. It adds to that:

- Additional top level / policy widgets.
- Component architecture (CORBA / Bonobo), embedding and document object model.
- Global configuration system (Gconf).
- Data storage using XML.
- Meta data and mime types support.
- Sound, image and printing support.
- Virtual file system access.

[6] http://www.gnome.org/
[7] Gnome = GNU Network Object Model Environment

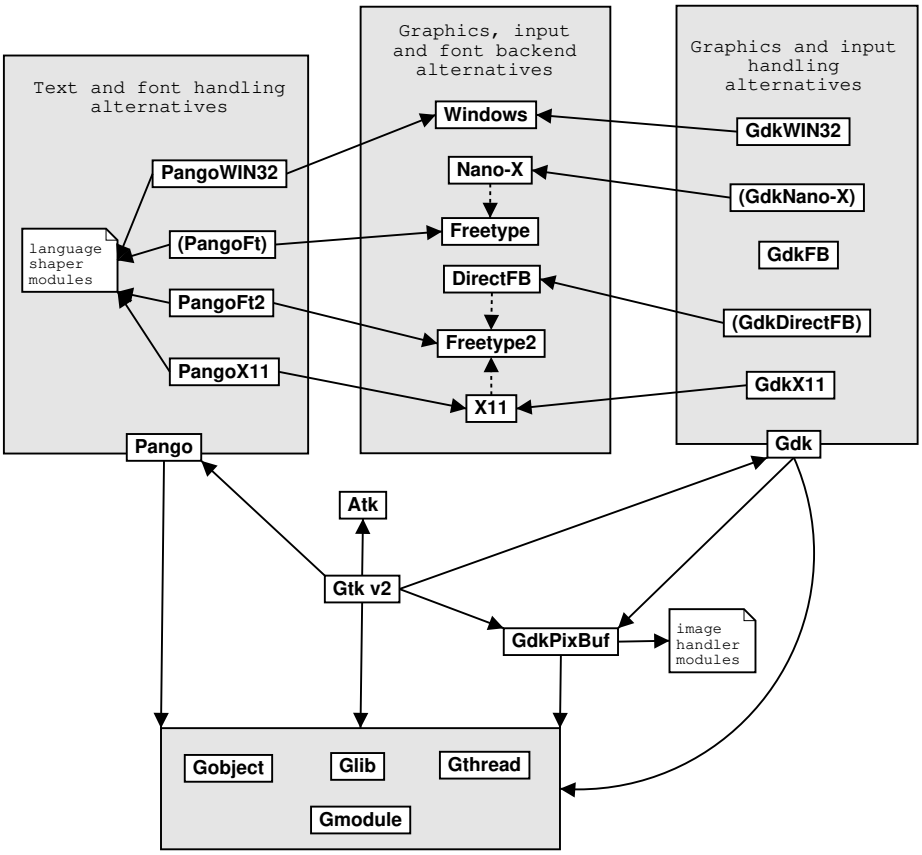


Figure 2: Gtk backend alternatives and dependencies

- Database access.
- X session management and a few different window managers.
- Desktop and office applications.
- Build and documentation environment.

Gnome is intended for desktop computers and it's too large for PDAs, but there might be some components that can be of use and adapted for PDAs with lots of space and RAM. Library interdependencies can make this tricky.

2.3 GPE

GPE, the *GPE Palmtop Environment* is a PDA programming environment based on top of the Familiar handhelds.org Linux distribution.

2.3.1 GPE components

GPE is currently composed of the following components and dependencies:

Familiar GPE components depend on some components provided by the Familiar distribution[10] and GPE uses the same Debian derived *ipkg* package management as Familiar.

Tiny-X A smaller version of the XFree86 X Window System v4.x.

Gtk An earlier and smaller 1.2 version of Gtk.

GPE-library A small library that implements a few PDA specific widgets, overrides some Gtk widgets and contains application setup code This is still in early development.

Matchbox A window manager implementing window policy that is geared towards PDAs. In this policy applications should have only one "main" window open at the time. The window size will be maximized, and all the other windows of the applications (dialogs, toolbars and other windows *marked as transient*) are linked to the main window. Current Matchbox doesn't yet support fullscreen mode.

uSQL A client-server version of the SQLite database and it's command line shell. Unlike the underlying SQLite, this is still in the early development.

Esd The Enlightened Sound Daemon. This is still in early development.

Applications There are some simple PIM applications, PDA configuration utilities and games using the GPE framework.

Project has produced some guidelines for PDA application development using the GPE framework. See`http://qpe.handhelds.org/`.

[10] http://familiar.handhelds.org/

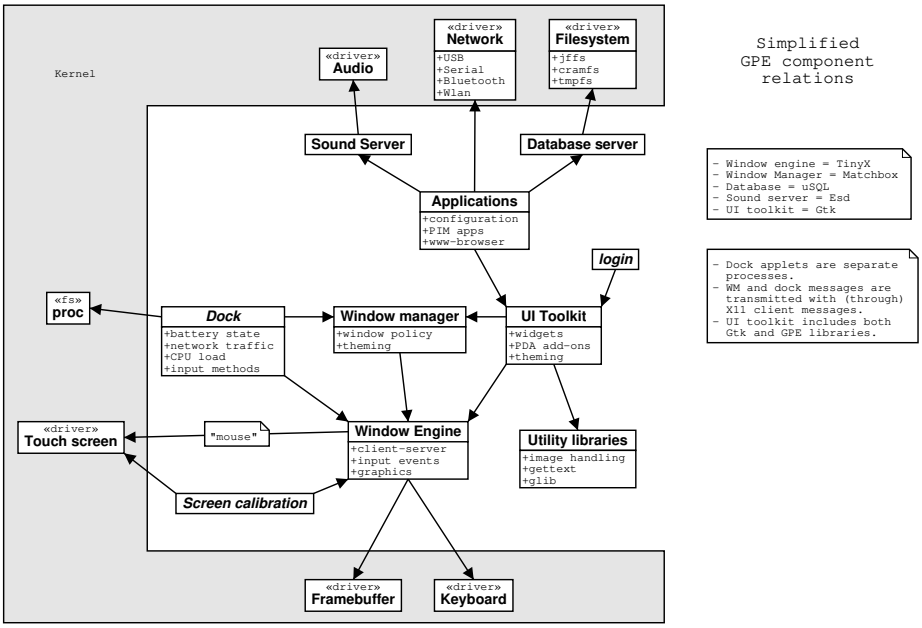


Figure 3: GPE dependency diagram.

2.3.2 GPE notes

- GPE components support both the portrait and landscape screen modes (XRANDR screen rotation extension).
- For some of the GPE applications, the interfaces are made using the Glade Gtk GUI builder v1 (see section 5.1) and the texts in the user interfaces are saved to gettext catalogs for localization. Some applications (e.g. Matchbox and clipboard) use X11 API instead of Gtk API.
  - Use of the older Gtk 1.2 version means that GPE applications don't support Unicode text strings and advanced international text layout.
  - Use of Gtk (and Glade) means that the GPE user interface is themable through Gtk theming mechanism (see section 3.4). The Matchbox window manager has it's own mechanism (like do the all the Gnome2 window managers) for theming.
  - Most of the Glade-1 files can be converted to Glade-2 with the *libglade-convert* python tool.
- GPE meta package depends on the following ipkg packages: gpe-calendar, gpe-todo, gpe-appmgr, gpe-login, gpe-dm, gpe-session-scripts, xkbd, matchbox, gtk-engines-xenophilia, xfonts-75dpi, gpe-what, gpe-edit, gpe-soundbite, keylaunch, esd, sfm, detect-stylus, xstroke, dillo, gpe-soundserver, gpe-bootsplash, gpe-kbd, gtk-theme-gpe, gpe-minicalc, gpe-terminal, xcalibrate, xfonts-ttf, ipaqscreen, update-menus.
  - Most of the GPE configuration utilities come from the Familiar as such and don't utilize the GPE framework and the GPE packages

itself have additional dependencies to other Familiar *ipkg* packages (e.g. xlibs, xserver, glibc) .

3 Features

3.1 Generic Gtk features

- Object oriented and implemented in C.
- Offers bindings for other languages (complete support for Ada, C++, Perl and Python).
- Drag and Drop (both Motif and X dnd protocols).
- Input methods (X11R6 XIM standard).
- Localization (GNU gettext and locale functions).
- Full Unicode level 3 internationalization.
- Thread safe by using *Glib* functions.
- Theming support.
- WIN32 support.

3.2 Unicode and font features

- Unicode 3 support (bi-directional text and languages) using *Pango*.
- Support GUI for directionality. Besides text, also widgets (e.g. check boxes) can be either left-to-right or right-to-left.
- *Pango* doesn't support vertical texts (in v1.0) Also, the currently supported X font system doesn't provide information required for *high* quality internationalized output (accent positioning, ligature and alternate glyph forms). Other toolkits are no better on this respect though.
- *Pango* has separate module for each language / font engine combination. Each font engine offers different capabilities for implementing more advances language display features (e.g. glyph composing for Thai).
- *Pango* reference implementation is based on X fonts, but the basic language shaper has also support for Xft (X with freetype), Freetype2 and WIN32.
- Freetype2 supports TrueType, Type1 (postscript), OpenType outline font and X11 PCF and Windows FNT bitmap font formats.
- Currently *pango* uses internally UTF-8, like rest of the Gtk. Switching into UCS-4 has been discussed.

3.3 Widgets

TODO: *Widget notes were originally for Gtk v1.2 and may not be fully updated to Gtk v2.*

Gtk has a rich set of widgets and it include all the commonly used widgets at least in some form. Here are some additional details about them:

Windows Following options can be toggled: borders, modality, resizability, sticking, destroy with parent. Following parameters can be changed: icon, title, size, position.

Containers There are horizontal and vertical box, horizontal and vertical pane, table, notebook (tabs), fixed co-ordinate and layout (with custom drawing) widgets, alignment and aspect ratio constraint widgets, frame and separator ornament container widgets.

Buttons Buttons can be either normal buttons, toggle buttons, check buttons or radio buttons. You can 'pack' other widgets (text, image) inside a button.

Label Labels can have multiple lines of text and the text can be justified, word wrapped or underlined (with an underline pattern).

Choice list Gtk has menu widgets. They can have tear-off items.

Combobox list Gtk has it.

Scrollbar Can be either vertical or horizontal. There are also range and scale widgets (with lots of options) for selecting and showing input values.

Progress bar Progress bar can work in 4 different orientations and either in continuous or discreet mode,

Control bar There are both scale and range widgets. Widget updates can be either continuous, discontinuous or delayed.

Menu These can be created manually from menu item, menu and menubar widgets or you can use *ItemFactory* that creates it from an array you provide.

Text widget Text widget can have multiple views on the same text, different font styles, text spacing and colors, wrapping, justification, Unicode and bidirectional text (not vertical).

Single line text input Supports cut and paste. Editability and visibility can be toggled.

Multiline text input Supports word wrap, cut and paste, different colors and fonts. Doesn't currently support horizontal scrollbars. Editability can be toggled,

Secret text input Single line text input with visibility off.

Popup dialog Gtk has a normal dialog and message dialog widgets. Dialog can be set to be modal.

Image support GtkImage widget supports all that Gdk supports. Gdk does all the required color etc. handling for images and it supports both client and server side images.

Animation GdkPixbufs handle issues related to animation[11].

Structured string input *No widget included for this into Gtk v1.2.*

Tabs Notebook widget tabs can be on any of the 4 edges. Tabs can also be hidden.

Listbox There's are normal listbox and multicolumn widgets. Latter can have titles on the columns.

Snaking listbox This might be implemented with multiple column listbox.

Tree There are normal tree and multicolumn tree widgets.

Color selector Color can be selected with RGB and HSV sliders or picking from HS-wheel / V-bar. Opacity of color can also be selected. *This doesn't fit into iPAQ screen.*

File selector dialog Gtk has two pane (directory and file panes) file selection dialog. *This doesn't fit into iPAQ screen.*

Font selector Font selector and preview. *This doesn't fit into iPAQ screen.*

Calendar Calendar widget has many options (week starts on Monday, show week numbers, show day names, whether to let user change month).

There was no **character map** widget but such should be available as a separate application or input methods.

Focus change order can be specified between widgets.

3.4 Theming

Gtk provides support for user-interface customization via *themes.* Without recompiling either Gtk or the application, a new look can be given to Gtk applications by installing a new theme.

Gtk themes are composed of following components:

Resource file Lists the styles, their attributes and key bindings and binds the styles to widgets. If no style is given for a widget or some style attributes are missing, widget inherits these values from it's parent class (see widget class hierarchy). The resource file format is documented here `http://developer.gnome.org/doc/API/gtk/gtk-resource-files.html`.

Styles Contain attributes used in drawing operations. Styles can include attributes for colors, line thickness, spacing, images and fonts for the widget and a pointer to a theme engine to use. Style attributes are *theme engine specific.*

[11] On PDA you might for efficiency reasons want to do video straight to the frame buffer instead of going through the widget set and window engine though.

Theme engine Is a shared library implementing functions for drawing shadowed boxes, frames, arrows, check-button indicators etc. These functions are listed in Gdk drawing API that is used by the widgets. See GtkStyle structure and class on this page `http://developer.gnome.org/doc/API/gtk/gtk-styles.html` for information on functions. Notes about theme engine implementations are here `http://www.gtk.org/~otaylor/gtk/2.0/theme-engines.html`.

Pixmaps Are image files referenced by the styles. These include widget backgrounds, stock icons and other images used by drawing functions implemented by the theme engine. Pixmap use is not supported by all the theme engines.

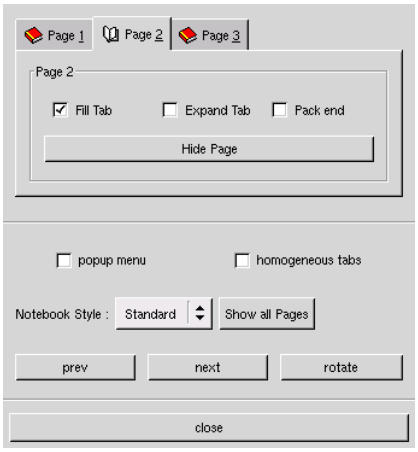
The default style setting are inside the Gtk code. Only other theme that Gtk source code installation provides is Emacs theme which will just override the default Gtk key bindings (i.e. theme can be provide only new key bindings while everything else comes from defaults).

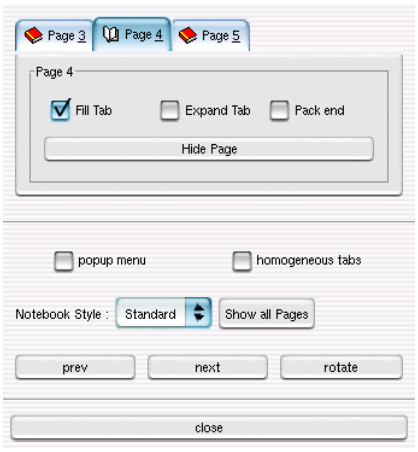
Typically Gtk themes reside in the `/usr/share/themes/<theme>/gtk/` directories. Theme engine libraries are usually distributed separately from theme resource files and pixmaps required by the styles as most themes use the same theme engines.

Here's a simple example of a pixmap style:

```
# "my-button" style, which overrides box drawing
# function with my own stretchable box image.
style "my-button" {
    xthickness = 2
    ythickness = 2
    engine "pixmap" {
        image    {
            function = BOX
            file         = "button.png"
            border     = { 2, 2, 2, 2 }
            stretch    = TRUE
        }
    }
}
# bind style to a widget class
class "GtkButton" style "my-button"
```

3.4.1 Changing theme





By default Gtk reads the global *gtkrc* and user's own *.gtkrc* resource files. In addition, Gtk will also read a locale specific version of each if they exist[12]. The global *locale specific* resource file typically just sets default fonts to use in that locale. Fonts can be specified with their X11 names or using the logical names *Pango* understands.

To use another theme, first you need to have the theme engine library and required pixmaps in the same directory as the resource file. They can also be in a directory you've specified in the *gtkrc* file or in GTK_PATH environment variable. Then you can include the theme resource file in your normal *~/.gtkrc* with *either* of these lines:

```
include "/path/to/theme/gtkrc"
theme "MyTheme"
```

If you want to set all the the resource files Gtk will load, you can set the GTK2_RC_FILES environment variable to point to them[13]. This way you could even have a separate theme for each application.

If you want all the running Gtk applications to change their theme, you edit an appropriate resource file and then send the applications an event that tells them to reload their resource files[14]. Gdk component will offer the event functions. There are also functions for applications to querying and changing application specific theme settings.

[12] With $LANG environment variable set to 'en_GB', locale version of *gtkrc* would be *gtkrc.en_GB*.

[13] For example KDE 3.0 in SuSE 8.0 sets GTK2_RC_FILES to value /etc/gtk/gtkrc:/home/user/.gtkrc:/home/user/.gtkrc-kde

[14] A v2 of the *gtk-theme-switch* program will do that for you.

3.4.2 Existing themes

Freshmeat theme site lists 14 themes and 2 theme engines for Gtk v2[15] and 400 themes for Gtk v1. It's possible to use multiple theme engines within a single theme.

To use most of these themes, you have to first install *gtk-engines* package from the Gnome project, because that offers the *pixbuf* engine used by most pixmap themes. It is compatible to the *pixmap* engine in the Gtk v1.

Gtk theme engine libraries are typically under 40KB and pixmap themes are about 300-400KB of PNG files. Themes that just specify colors and key bindings for the default looks may be only a few kilobytes.

In addition to widget set theme, comprehensive themes could also contain theming data for the window manager (backgrounds & colors), panel (icons & colors), media player (skin pack), browser (XUL definition & data files) etc.

3.4.3 Theming limitations

Theme engines, and therefore the themes using them, can change the internal *layouts* of widgets *only* by changing internal attributes and drawing operations of the widgets. An example of this would be moving scrollbar arrows together. Currently there's no public API for doing this kind of things.

Only way to do above safely when user can change themes, is to make sure that this kind of a theme engine is *never* unloaded because otherwise the changed widget pointers would be invalid. Another shortcoming is that if the new style doesn't overwrite also the modified widget internals, user gets a mixed-up GUI where some widget parts remain similar to an earlier theme. If system will have only one theme, this is acceptable.

If modifications are significant, it may be wiser to write new widget conforming to the old widget API and contribute the widget back to Gtk project.

The application layouts can only be changed by changing the applications itself, which is easy if application interfaces are done using Glade. It's possible to do minor global modifications by poking with the internals of layout manager widgets, but this is not advised.

3.5 Modularity

The supplied build environment doesn't support smaller toolkit configurations with less widgets and (in case of Pango) fewer language engines. Those configurations have to be built manually. It is possible to automate this later on to better track toolkit changes, but the initial effort to build this well would be fairly large.

[15] http://themes.freshmeat.net/browse/958/?topic_id=958

3.6 Event Handling

For handling events Gtk uses *signals* instead of call backs. The difference between signals and events is that signals have nothing to do with input streams, they are just machinery for handling call backs when something changes. For more information on this, see "Programming Gtk" on page 22.

3.7 Error handling

Gtk widgets use extensive Glib allocation and list functions. Glib uses internally it's own allocation routines. In case allocation fails, these functions *log*() it with LOG_LEVEL_ERROR (each of the Gtk components has it's own Glib logging 'domain') which Glib logging functions interpret as fatal and *abort*() the program. Calling *abort*() will terminate the program unless signal handler for *SIGBRT* is installed.

Glib allocation routines have provisions for using external memory debugging and profiling tools.

Other system calls Gtk code seems to check normally.

4 Building Gtk

4.1 Compiling and installing

Gtk is build using the *configure* script and GNU *make*. Building Gtk requires *pkgconfig* tool. Creating the documentation requires the *gtk-doc* package and SGML tools.

Each Gtk component contains *spec file* that will be installed as part of the component installation. Configure script will query from the pkgconfig tool the options required for using the underlying components in compiling and linking the currently configured component.

Configuring required for cross compiling Gtk components is problematic as it's configure scripts have many places which don't work with cross compiling as the tests in the scripts try to run cross-compiled test programs natively.

Libraries are compiled using gcc. Other compilers were not tested.

Sources are available from the Gtk FTP site `ftp://ftp.gtk.org/pub/gtk/v2.0/`.

4.2 Binary sizes

Building and size estimates were done using the following rules:

- Gtk v2.0.6 sources were used for a compilation on a x86 SuSE 8.0 Linux system with linux kernel 2.4.18 (unaccelerated VESA framebuffer driver), glibc-2.2.5 and XFree86-4.2.

- Glib and Gtk configuring was done with thread support and the *–enable-debug=minimum* configure option.
- Compilation of Gtk libraries was done with the *gcc-2.95.3* compiler using the *-Os* optimization option. DirectFB was compiled with the *-O2 -mpentium* options.
- Produced dynamic libraries were stripped with *strip* using the '*-R=.notes -R=.comment*' options.
- Sizes are reported as Kilobytes (1024 bytes).

Gtk libraries that *gtk-demo* application depends are following.

Name:	Size:
libgtk-x11	2099k
libpangoxft	123k
libpangox	45k
libpango	193k
libgdk-x11	349k
libgdk_pixbuf	70k
libatk	85k
libgobject	221k
libgthread	14k
libgmodule	10k
libglib	408k
sum:	**3617**k
libXft	91k
libXrender	17k
libfreetype	250k
libXext	51k
libX11	783k
libdl	14k
libm	184k
glibc	1361k
ld-linux	97k
sum:	**2848**k

Above are dependencies and their sizes for Gtk on X11, below at left for Gtk on framebuffer and at right for Gtk on DirectFB.

Name:	Size:
libgtk-linux-fb	2085k
libpangoft2	184k
libpango	45k
libgdk-linux-fb	382k
libgdk_pixbuf	70k
libatk	85k
libgobject	221k
libgthread	14k
libgmodule	10k
libglib	408k
sum:	**3504**k
libfreetype	250k
libdl	14k
libm	184k
glibc	1361k
ld-linux	97k
sum:	**1906**k

Name:	Size:
libgtk-directfb	2064k
libpangoft2	184k
libpango	45k
libgdk-directfb	312k
libgdk_pixbuf	70k
libatk	85k
libgobject	221k
libgthread	14k
libgmodule	10k
libglib	408k
sum:	**3413**k
libfreetype	250k
libdirectfb	205k
libdl	14k
libm	184k
glibc	1361k
ld-linux	97k
sum:	**2111**k

For the X11 version you will need also X server. DirectFB will load at runtime additional libraries to handle different input and graphics devices and to interface to different font, image and video formats when needed. Input drivers are about 10KB each, graphics drivers about 20KB and interface libraries about 15KB each + the libraries they depend on (flash, jpeg, freetype etc).

4.3 Runtime loaded components

In addition to these, Gtk components can dynamically load some helper libraries when they are needed.

Gdk_pixbuf can load different *image format loaders* which may depend on other libraries (libjpeg, libpng+libz, libtiff = **652**k), Gdk can load dynamically different *input method handlers* and Pango can load it's *language shapers*.

There are currently available 36 of these helper libraries; 260k worth of language shapers, 140k worth of input methods (for X11) and 200k worth of image loaders. In all **600**k of helper libraries to choose from.

Also, pango component will load fonts and theming will load theming engine libraries and theme images when needed.

4.4 Runtime impressions

Gtk test application widgets worked very well. With the largest *testgtk* application the memory consumption on X11 raised from initial 4MB to 18MB during the testing of all the available Gtk widgets. It's tests included some *very* large viewports and grids.

On framebuffer the test application redraws flickered a bit and when something was moved, the background painting was very noticable. It also uses more memory but less than X11 version + X server. In general the X11 version of Gtk felt much slicker.

DirectFB version was on par with X11 on slickness of the widget use and background handling. Like Gtk framebuffer version, it used the optional window decoration code *included into Gtk* for titlebars with which the application windows could be moved around. DirectFB doesn't implement some more complex drawing operations that only a small subset of Gtk widgets need (e.g. arcs are missing).

Neither GtkFB nor DirectFB can run multiple applications on the same screen yet and with both the applications have to be run as 'root'.

5 Using Gtk

5.1 User interface builder

Glade is a RAD tool to enable quick & easy development of user interfaces for the GTK+ toolkit and the GNOME desktop environment. It also contains built-in support for generating the C source code needed to recreate the interfaces. All you need to do is provide the code for the UI call backs.

Gnome version has additional panel for Gnome widgets and builtin documentation browser.

5.1.1 Dependencies and building

The user interfaces designed in Glade are stored in the well-known XML format, enabling easy integration with external tools. Several tools are already available which can turn the XML files into source code in other languages such as C++, Perl and Python. Other tools such as libglade can load the XML files and *create the interfaces at runtime*. Use of runtime interfaces will add *libglade* and *libxml* dependencies to the application. Other Glade applications have just added *libXi* dependency (30k on x86).

To compile programs output by Glade, you need GNU autoconf, automake, make and a full Gtk installation (includes some m4 macros). If you've selected gettext support, you will need also gettext and libintl. Glade creates directory structure, UI description file, gettext files, configure scripts and UI source code for the application.

5.1.2 Internationalization

Glade supports GNU gettext message catalogs and provides 'configure' and locale code for them. You will also need message catalog software for extracting the strings into catalog and translating the message catalogs.

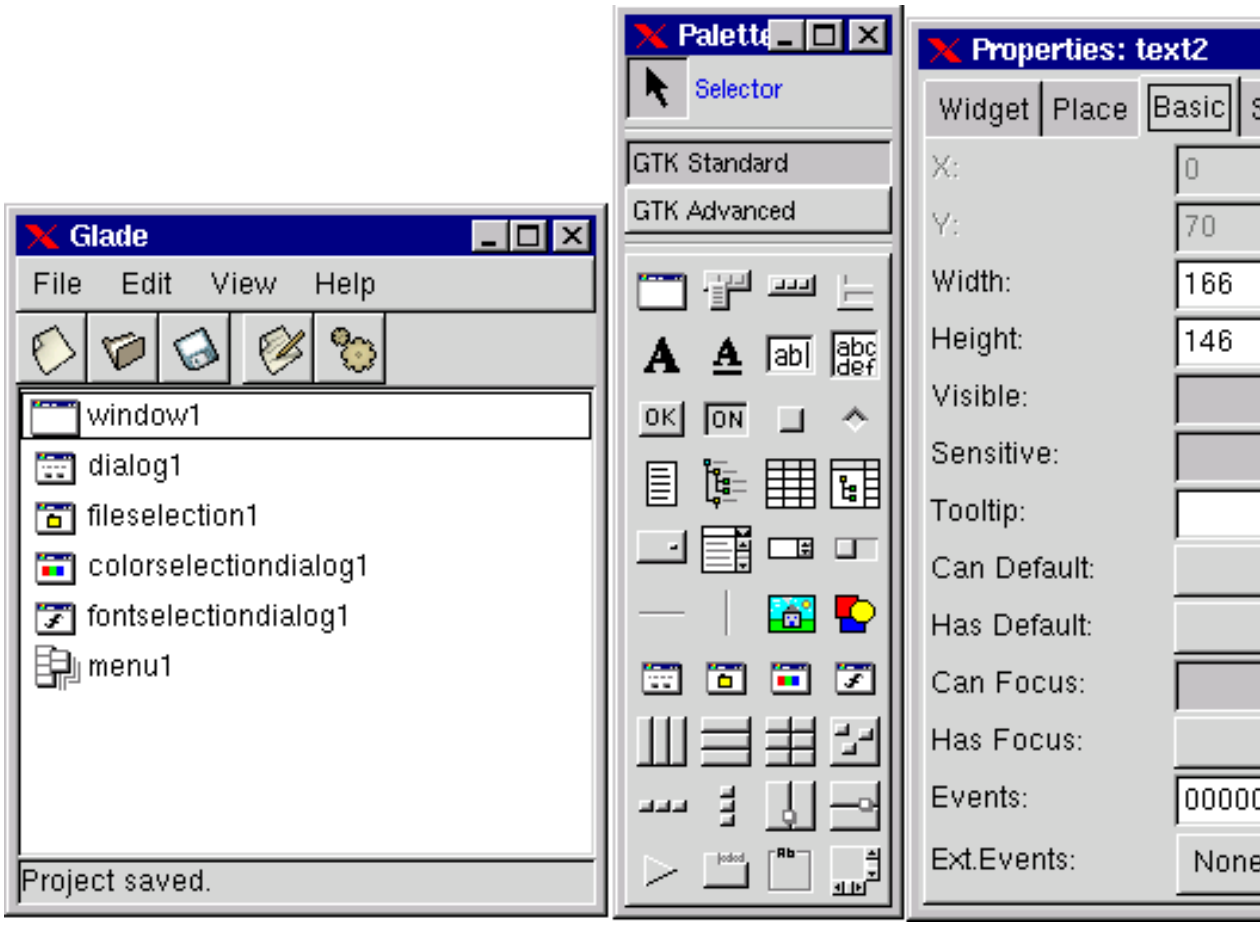


Figure 4: Glade project, widget palette and widget properties windows

5.1.3 Documentation and license

Documentation includes a FAQ and an unfinished User Guide. In the gnome version of Glade, these are accessible within the program. There are three mailing lists dedicated to Glade: glade-announce@ximian.com, glade-users@ximian.com and glade-devel@ximian.com.

Glade itself has GPL license but it's the documentation says that "you are free to use whatever license you like for the source code generated by Glade".

5.2 Programming Gtk

The general steps to creating a Gtk widget in C are:

1. Use on of the *gtk_ *_ new* - one of various functions to create a new widget.
2. Connect all signals and events we wish to use to the appropriate handlers.
3. Set the attributes of the widget.
4. Pack the widget into a container using the appropriate call such as *gtk_ container_ add*() or *gtk_ box_ pack_ start*().
5. Use *gtk_ widget_ show*() to show the widget.

Although Gtk is written in C, it's programmed in object oriented manner:

- Widgets are inheritable.
- Events are connected to widgets using signals[16], like this:
  `gtk_signal_connect(GTK_OBJECT(button),"clicked",`
  `GTK_SIGNAL_FUNC(callback_function), callback_data);`
- For operations like signal connecting, objects (C structures) and functions have to be "casted" into their base type. Available casting operation macros are:
  `GTK_WIDGET(widget)`
  `GTK_OBJECT(object)`
  `GTK_SIGNAL_FUNC(function)`
  `GTK_CONTAINER(container)`
  `GTK_WINDOW(window)`
  `GTK_BOX(box)`

Gtk v2 widget hierarchy[17] is following:

```
GObject
    GtkAccelGroup
    GtkObject
        GtkWidget
```

[16] In Gtk v2 the underlying object signaling mechanism moved from Gtk to Glib package.
[17] See "Object hierarchy" at: `http://developer.gnome.org/doc/API/2.0/gtk/index.html`.

```
GtkMisc
    GtkLabel
        GtkAccelLabel
        GtkTipsQuery
    GtkArrow
    GtkImage
    GtkPixmap
GtkContainer
    GtkBin
        GtkAlignment
        GtkFrame
            GtkAspectFrame
        GtkButton
            GtkToggleButton
                GtkCheckButton
                    GtkRadioButton
            GtkOptionMenu
        GtkItem
            GtkMenuItem
                GtkCheckMenuItem
                    GtkRadioMenuItem
                GtkImageMenuItem
                GtkSeparatorMenuItem
                GtkTearoffMenuItem
            GtkListItem
            GtkTreeItem
        GtkWindow
            GtkDialog
                GtkColorSelectionDialog
                GtkFileSelection
                GtkFontSelectionDialog
                GtkInputDialog
                GtkMessageDialog
            GtkPlug
        GtkEventBox
        GtkHandleBox
        GtkScrolledWindow
        GtkViewport
    GtkBox
        GtkButtonBox
            GtkHButtonBox
            GtkVButtonBox
        GtkVBox
            GtkColorSelection
            GtkFontSelection
            GtkGammaCurve
        GtkHBox
            GtkCombo
            GtkStatusbar
    GtkCList
```

```
                GtkCTree
            GtkFixed
            GtkPaned
                GtkHPaned
                GtkVPaned
            GtkLayout
            GtkList
            GtkMenuShell
                GtkMenuBar
                GtkMenu
            GtkNotebook
            GtkSocket
            GtkTable
            GtkTextView
            GtkToolbar
            GtkTree
            GtkTreeView
        GtkCalendar
        GtkDrawingArea
            GtkCurve
        GtkEntry
            GtkSpinButton
        GtkRuler
            GtkHRuler
            GtkVRuler
        GtkRange
            GtkScale
                GtkHScale
                GtkVScale
            GtkScrollbar
                GtkHScrollbar
                GtkVScrollbar
        GtkSeparator
            GtkHSeparator
            GtkVSeparator
        GtkInvisible
        GtkOldEditable
            GtkText
        GtkPreview
        GtkProgress
            GtkProgressBar
    GtkAdjustment
    GtkCellRenderer
        GtkCellRendererPixbuf
        GtkCellRendererText
        GtkCellRendererToggle
    GtkItemFactory
    GtkTooltips
    GtkTreeViewColumn
AtkObject
```

```
    GtkAccessible
GtkIconFactory
GtkIMContext
    GtkIMContextSimple
    GtkIMMulticontext
GtkListStore
GtkRcStyle
GtkSettings
GtkSizeGroup
GtkStyle
GtkTextBuffer
GtkTextChildAnchor
GtkTextMark
GtkTextTag
GtkTextTagTable
GtkTreeModelSort
GtkTreeSelection
GtkTreeStore
GtkWindowGroup
GdkDragContext
GdkPixbuf
GdkDrawable
    GdkPixmap
GdkImage
GdkPixbufAnimation
GdkDevice
```

6 Documentation

6.1 Online documentation

Gtk has comprehensive developer documentation available online:

- FAQ `http://www.gtk.org/faq/`.
- Large and comprehensive programming tutorial `http://developer.gnome.org/arch/doc/devel-doc.html`.
- Full API references for *Glib, GObject, Pango, GdkPixbuf, Gdk* and *Gtk*. Documentation is available here `http://www.gtk.org/api/`.
- An overall architecture documentation to Gtk is included into the Gnome project documentation here `http://developer.gnome.org/arch/`.

Tutorial is available in it's source format here `ftp://ftp.gtk.org/pub/gtk/tutorial/` and API references are included with the package sources.

Documents are maintained as *linuxdoc SGML* which can be converted into HTML, LaTeX, PDF and text formats.

6.2 Gtk books

There are two books published on Gtk and Gnome application development:

- "Developing Linux Applications with GTK+ and Gdk"[18] by Eric Harlow.
- "GTK+/GNOME Application Development"[19] by Havoc Pennington. The free version of the book is here `http://developer.gnome.org/doc/GGAD/`.

7 Community

The information in this section is from the end of 2001.

7.1 Organizational support

Gtk development seems mainly backed by *RedHat*[20] as many prominent Gtk developers (like Owen Taylor and Havoc Pennington) are employed by it. *Sun Microsystems* has contributed test code to Gtk and is planning to help in localization.

Gnome development is backed by *Ximian*[21] (makes Gnome desktop applications for business use), *Sun Microsystems* (Open Office integration and UI testing[22]) and in future, possibly also by *HP*. Sun and HP both intend to offer Gnome as an alternative for CDE on their Unix OSes.

There's also a nonprofit *Gnome Foundation*[23] organization founded to fund and to help direct the future course of Gnome. It's Advisory Board is currently composed of following companies: Borland, Compaq, HP, IBM, Sun Microsystems, Object Management Group, Debian Project, Free Software Foundation, Gnumatic, MandrakeSoft, Red Flag Linux, Red Hat, TurboLinux, VA Linux, Ximian.

Almost all Linux distributions support both Gnome/Gtk and KDE/Qt nowadays.

7.2 Mailing lists

There are several mailing lists devoted to Gtk discussion:

- **gtk-list@gnome.org**: main list.
- **gtk-app-devel-list@gnome.org** : application development list.
- **gtk-devel-list@gnome.org:** code discussion list.

[18] The ISBN is 0-7357-0021-4.
[19] The ISBN is 0-7357-0078-8.
[20] http://www.redhat.com/
[21] http://www.ximian.com/
[22] http://developer.gnome.org/projects/gup/ut1_report/
[23] http://www.gnome.org/faqs/gnome-foundation-faq/

- **gtk-doc-list@gnome.org:** documentation list.
- **gtk-i18n-list@gnome.org:** Internationalization and localization list. Discussion are about Pango implementation, help requests, patches, announcements about new Pango modules or versions.

Pango list has on average one mail a day and other mailing lists have all together about 40-80 new mails per day.

7.3 Bugs

Bugs reported to the mailing lists are usually answered within the same day.

Bugs are reported to Bugzilla `http://bugzilla.gnome.org/`. There are separate categories for each of the libraries and most of the larger applications.

7.4 Testing and quality control

In Open Source projects quality control is mostly done by "release early and often" strategy on the *development branch*.

With graphics user interface toolkits the trouble is that unit and regression tests work by checking that given input produces required output. In user interfaces the input comes from user and output is something visual.

This means that quality control would be more like test teams and their checklists[24] and examples of correct looking visuals. *For underlying libraries and components, the tests can be automated:*

- In Gtk v2 there's test code for each of the libraries, but it's not complete. There's no build target for testing and it's not documented yet.

At the moment release stability is based on main developers feel of it (there are no formal test teams within the project, outside of it there are in companies, but they are not public) and the number of outstanding bugs in the project Bugzilla database.

8 Terminology

Gtk In this document refers to any Gtk version.

Gtk+ Name of the current Gtk version on Gtk documentation. After Gtk acquired signal mechanism and widgets became inheritable a '+' was added after it's name. This convention is not used in this documentation.

[24] E.g. open file selector and resize it both larger and smaller. Does it crash, are there any redraw errors?

Part II

Appendixes

9 Document distribution

10 Changelog

2002-08-27 Started the document based on the Gtk evaluation from the last fall. Removed Gtk v1 specific parts, old information for the development version of Gtk v2 and cross-compilation appendixes. Partly rewrote the section on what different Gtk v2 components do and added target notes.

2002-08-29 Build Gtk and finished the Gtk component description updates. Rewrote Gtk building and binary sizes sections.

2002-08-30 Tested a couple of Gtk themes and wrote the theming section.

2002-09-03 Branched the feasibility study to it's own document. Spellchecked the document and improved the component sections.

2002-09-04 Updated theming sections, built and tested DirectFB and Gtk for it and for plain linux framebuffer.

2002-09-05 Rewrote the building section to include information on GtkFB and Gtk / DirectFB.

2002-09-10 Added a Pango rendering pipeline diagram based on notes by Simo Piiroinen.

2002-09-18 Added more information on GPE.

2002-09-26 Updated the theming section and widget hierarchy.

2002-09-30 Simplified the "Gtk backend and dependency" diagram.

2002-10-14 Prepared the document for public release.