The General Toolkit

Version 1.0
May 1996

by Peter Mattis

The General Toolkit

1 Copying

GTK is *free*; this means that everyone is free to use it and free to redestribute it on a free basis. GTK is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of GTK that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of GTK, that you receive source code or else can get it if you want it, that you can change GTK or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of GTK, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for my own protection, we must make certain that everyone finds out that there is no warranty for GTK. If GTK is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will no reflect on our reputation.

The precise conditions of the licenses for GTK are found in the General Public Licenses that accompanies it.

2 What is GTK?

GTK is a library for creating user interfaces similar to the Motif "look and feel". It is designed to be small and efficient, but still flexible enough to allow the programmer freedom in the interfaces he/she creates. GTK allows the programmer to use a variety of standard user interface objects such as push, radio and check buttons, menus, lists and frames. It also has several "container" objects which can be used to control the layout of user interface elements such as horizontal and vertical boxes and tables.

3 Initialization and Exit

Initializing GTK is easy. Simply call `gtk_init` passing in the *argc* and *argv* parameters. Exit is similarly easy. Just call `gdk_exit`. Note that `gtk_init` will call `gdk_init`. Therefore, explicit initialization of GDK is not necessary by the programmer.

`void` **gtk_init** (`int` **argc*, `char` ***argv*) Function

Initialize the GTK library and has the side effect of initializing the GDK library. The arguments *argc* and *argv* are scanned and any arguments that GTK recognizes are handled and removed. The *argc* and *argv* parameters are the values passed to `main` upon program invocation.

`void` **gtk_exit** (`int` *errorcode*) Function

Exit GTK and perform any necessary cleanup. `gtk_exit` will call `gdk_exit` when it is finished cleaning up the GTK internals. The parameter *errorcode* will be passed to `gdk_exit`.

```
int
main (int argc, char *argv[])
{
  /* Initialize GTK. */
  gtk_init (&argc, &argv);

  /* Exit from GTK...this call will never return. */
  gtk_exit (0);

  /* Keep compiler from issuing a warning */
  return 0;
}
```

4 Widget Overview

Widgets are the general term used by GTK for user interface objects. A widget describes a certain interface that all user interface objects conform to. This interface allows a uniform method for dealing with operations common to all objects such as hiding and showing of an object.

The common interface that widgets must adhere to is described by the GtkWidget and GtkWidgetFunctions structure. For the purposes of using GTK these structures can be considered read-only and, for the most part, opaque. (The few exceptions are the need to examine and use the "type", "style" and "window" fields on occasion).

All widget creation routines in GTK return pointers to GtkWidget structs. In reality, all widget creation routines create structures that can be viewed as equivalent to the GtkWidget structure, but often have contain additional information.

The programmer can perform several operations with a widget. Either one of the basic widget operations can be invoked or one of the widget specific operations can be invoked. The basic widget operations are described below while the widget specific operations are described later in the manual.

`void` **gtk_widget_destroy** (`GtkWidget *`*widget*) Function

Destroys the specified widgets and any children it may have. All memory associated with the widget is released.

`void` **gtk_widget_show** (`GtkWidget *`*widget*) Function

Makes the specified widget visible. This has the effect of setting a flag within the widget specifying it as visible and notifying the widget's parent of its new state. If the parent is visible then sometime after this call has been made the widget will be realized and mapped. Note: it is not safe to assume the widget has been realized and mapped immediately following this call. A visible widget participates in geometry management.

`void` **gtk_widget_hide** (`GtkWidget *`*widget*) Function

Makes the specified widget invisible. This has the effect of unsetting a flag within the widget specifying it as visible and notifying the widget's parent of its new state. Sometime after this call has been made the widget will be unmapped. An invisible widget does not participate in geometry management.

`void` **gtk_widget_map** (`GtkWidget *`*widget*) Function

Maps a widget on screen.

`void` **gtk_widget_unmap** (`GtkWidget *`*widget*) Function

Unmaps a widget from the screen.

`void` **gtk_widget_realize** (`GtkWidget *`*widget*) Function

Realizes a widgets window. Realization consists of creating a widgets window and any other initialization needed before the widget is mapped.

`void` **gtk_widget_draw** (`GtkWidget *`*widget*`, GdkRectangle *`*area*`, gint` *is_expose*) Function

Draw the specified widget. The *area* and *is_expose* parameters are used by widget event code when an expose event arrives. For the purposes of redrawing a widget these parameters can take the values "NULL" and "FALSE" respectively.

`void` **gtk_widget_draw_focus** (`GtkWidget *`*widget*) Function

Draw the specified widgets focus highlight. If the widget has the focus then a 1 pixel wide highlight will be drawn around the widget. If the widget does not have the focus then a 1 pixel widget highlight will be drawn in the background color around the widget (effectively erasing any previous highlight).

`gint` **gtk_widget_event** (`GtkWidget *`*widget*`, GdkEvent *`*event*) Function

Send an event to a widget.

`void` **gtk_widget_size_request** (`GtkWidget *`*widget*`, GtkRequisition *`*requisition*) Function

Query a widget as to its desired size.

`void` **gtk_widget_size_allocate** (`GtkWidget *`*widget*`, GtkAllocation *`*allocation*) Function

Allocate space for a widget.

`gint` **gtk_widget_is_child** (`GtkWidget *`*widget*`, GtkWidget *`*child*) Function

Query a widget to determine if *child* is a child of *widget*.

`void` **gtk_widget_is_immediate_child** (`GtkWidget *`*widget*`, GtkWidget *`*child*) Function

Query a widget to determine if *child* is an immediate child of *widget*.

`void` **gtk_widget_locate** (`GtkWidget *`*widget*`, GtkWidget **`*child*`, gint` *x*`, gint` *y*) Function

Locates the child widget located at position *x*, *y*. The child widget is returned in the parameter *child*.

`void` **gtk_widget_activate** (`GtkWidget *`*widget*) Function

Activate a widget.

`void` **gtk_widget_set_state** (`GtkWidget *`*widget*`, GtkStateType` *state*) Function

Set a widgets state.

`void` **gtk_widget_install_accelerator** (`GtkWidget *`*widget*`, gchar` *accelerator_key*`, guint8` *accelerator_mods*`, gint` *global*) Function

Install a keyboard accelerator. When *accelerator_key* is invoked with the modifier keys *accelerator_keys*, then the specified widget will be "activated" via `gtk_widget_activate`. (It therefore makes little sense to add a keyboard accelerator for a widget which does not have an activate routine). If the *global*

flag is "TRUE" the accelerator will be installed in the global accelerator table making it a valid accelerator for any window. If the *global* flag is "FALSE" the accelerator is installed in specified widgets toplevel window making it a valid accelerator only when the pointer is in the toplevel window the widget is in. Note: any existing accelerator using the combination of *accelerator_key* and *accelerator_mods* will get removed.

`void` **gtk_widget_remove_accelerator** (`GtkWidget *`*widget*, `gint` *global*) Function

Remove a keyboard accelerator.

`void` **gtk_widget_grab_focus** (`GtkWidget *`*widget*) Function

Grab the focus for a widget.

`gint` **gtk_widget_intersect** (`GtkWidget *`*widget*, `GdkRectangle *`*area*, `GdkRectangle *`*dest*) Function

Determine the intersection between a widget and the rectangle *area*. The resulting intersection returned in *dest* is valid only if `gtk_widget_intersect` returns "TRUE".

`void` **gtk_widget_reparent** (`GtkWidget *`*widget*, `GtkWidget *`*new_parent*) Function

Change a widgets parent.

5 Container Overview

6 Style Overview

7 Data and Observer Overview

8 Using timers

9 Using accelerators

10 Grabbing events

11 Creating Alignment Containers

12 Creating Box Containers

13 Creating Buttons

14 Creating Drawing Areas

15 Creating Frame Containers

16 Creating Lists and Listboxes

17 Creating Menus

18 Creating Labels, Images and Pixmaps

19 Creating Rulers

20 Creating Scales

21 Creating Scrolled Areas and Windows

22 Creating Scrollbars

23 Creating Table Containers

24 Creating Text Entry Fields

25 Creating Windows

26 Examples

Function Index

(Index is nonexistent)

Concept Index

(Index is nonexistent)

Short Contents

Table of Contents