The General Drawing Kit

Version 1.0
May 1996

by Peter Mattis

The General Drawing Kit

1 Copying

GDK is *free*; this means that everyone is free to use it and free to redestribute it on a free basis. GDK 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 GDK that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of GDK, that you receive source code or else can get it if you want it, that you can change GDK 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 GDK, 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 GDK. If GDK 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 GDK are found in the General Public Licenses that accompanies it.

2 What is GDK?

GDK is designed as a wrapper library that lies on top of Xlib. It performs many common and desired operations for a programmer instead of the programmer having to explicitly ask for such functionality from Xlib directly. For example, GDK provides a common interface to both regular and shared memory XImage types. By doing so, an application can nearly transparently use the fastest image type available. GDK also provides routines for determining the best available color depth and the best available visual which is not always the default visual for a screen.

3 Initialization and exit

Initializing GDK is easy. Simply call `gdk_init` passing in the *argc* and *argv* parameters. Exit is similarly easy. Just call `gdk_exit`.

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

Initializes the GDK library. The arguments *argc* and *argv* are scanned and any arguments that GDK recognizes are handled and removed. The *argc* and *argv* parameters are the values passed to `main` upon program invocation.

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

Exit GDK and perform any necessary cleanup. `gdk_exit` will call the systems `exit` function passing *errorcode* as the parameter.

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

  /* Exit from GDK...this call will never return. */
  gdk_exit (0);

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

4 Event handling

Events are the means by which GDK lets the programmer know of user interaction. An event is normally a button or key press or some other indirect user action, such as a the mouse cursor entering or leaving a window.

There exist only a few functions for getting events and event information. These are `gdk_events_pending`, `gdk_event_get`, `gdk_events_record` and `gdk_events_playback`. The latter two functions are useful for automatic testing of a software package and should normally not be needed in a program.

`gint` **gdk_events_pending** (`void`) Function

Returns the number of events pending on the event queue.

`gint` **gdk_event_get** (`GdkEvent` **event*) Function

Return the next available event in the *event* structure. `gdk_event_get` will return `TRUE` on success and `FALSE` on failure. Success and failure is determined by whether an event arrived before the timeout period expired.

`void` **gdk_events_record** (`char` **filename*) Function

Turn on recording of events. User events and certain system events will be saved in the file named by the variable *filename*. This stream of events can later be played back and "should" produce the same results as when the original events were handled. However, the programmer does need to be careful in that the state of the program must be the same when `gdk_events_record` is called and when `gdk_events_playback` is called. For this reason, `gdk_events_record` is normally not called directly, but is instead invoked indirectly by specifying the "-record" command line option.

`void` **gdk_events_playback** (`char` **filename*) Function

Start playback of events from a file. (See the above description of `gdk_events_record`). Normally this function is not called directly but is invoked by the "-playback" command line option.

`void` **gdk_events_stop** (`void`) Function

Stop recording and playback of events.

```
void
handle_event ()
{
  GdkEvent event;

  if (gdk_event_get (&event))
    {
      switch (event.type)
       {
         ...
       }
    }
}
```

5 Understanding and using visuals

6 Creating and using windows

7 Creating and modifying GCs

8 Creating pixmaps

9 Creating images

10 Specifying color

11 Creating Fonts

12 Drawing Commands

13 Using extended devices

14 Other stuff

15 Using GDK

Variable Index

(Index is nonexistent)

Concept Index

(Index is nonexistent)

Short Contents

Table of Contents