The EFL Cookbook

Various
Edited by Ben technikolor Rockwood

The EFL Cookbook

by Various and Ben technikolor Rockwood

Stuff.

Table of Contents

List of Examples

Chapter 1. Introduction

This book.....

More intro.

Chapter 2. Imlib2

Imlib2 provides a powerful engine for image manipulation and rendering.

Chapter 3. EVAS

Evas is a hardware-accelerated canvas API for X-Windows that can draw anti-aliased text, smooth super and sub-sampled images, alpha-blend, as well as drop down to using normal X11 primitives such as pixmaps, lines and rectangles for speed if your CPU or graphics hardware are too slow.

Evas abstracts any need to know much about what the characteristics of your XServer's display are, what depth or what magic visuals etc, it has. The most you need to tell Evas is how many colors (at a maximum) to use if the display is not a truecolor display. By default it is suggested to use 216 colors (as this equates to a 6x6x6 color cube - exactly the same color cube Netscape, Mozilla, gdkrgb etc. use so colors will be shared). If Evas can't allocate enough colors it keeps reducing the size of the color cube until it reaches plain black and white. This way, it can display on anything from a black and white only terminal to 16 color VGA to 256 color and all the way up through 15, 16, 24 and 32bit color. Here are some screen shots of a demo Evas application to show the rendering output in different situations

Recipe: Key Binds, using EVAS Key Events

This is my recipe!

Chapter 4. Ecore

Ecore provides a powerful event handling and modularized abstraction layer which ties and bind your applications various components together in a nearly seemless manner.

Recipe: Ecore Config Introduction

dan sinclair <zero@perplexity.org>

The Ecore_Config module provides the programmer with a very simple way to setup configuration files for their program. This recipe will give an example of how to integrate the beginnings of Ecore_Config into your program and use it to get configuration data.

Example 4.1. Simple Ecore_Config program

```
#include <Ecore_Config.h>

int main(int argc, char ** argv) {
    int i;
    float j;
    char *str;

    if (ecore_config_init("foo") != 0) {
        printf("Cannot init Ecore_Config");
        return 1;
    }

    ecore_config_default_int("/int_example", 1);
    ecore_config_default_string("/this/is/a/string/example", "String");
    ecore_config_default_float("/float/example", 2.22);

    ecore_config_load();

    i = ecore_config_get_int("/int_example");
    str = ecore_config_get_string("/this/is/a/string/example");
    j = ecore_config_get_float("/float/example");

    printf("str is (%s)\n", str);
    printf("i is (%d)\n", i);
    printf("j is (%f)\n", j);

    free(str);

    ecore_config_exit();
    return 0;
}
```

As you can see from this example the basic usage of Ecore_Config is simple. The system is initialized with a call to ecore_config_init(PROGRAM_NAME). The program name setting control where Ecore_Config will look for your configuration database. The directory and file name are: ~/.e/apps/PROGRAM_NAME/config.db.

For each configuration variable you are getting from Ecore_Config, you can assign a default value in the case that the user does not have a config.db file. The defaults are assigned with the ecore_config_default_* where * is one of the Ecore_Config types. The first parameter is the key under

which this is to be accessed. These keys must be unique over your program. The value passed is of the type appropriated for this call.

The ecore_config_load call will read the values from the config.db file into Ecore_Config. After which we can access the files with the ecore_config_get_* methods (again * is the type of data desired). These routines take the key name for this item and return the value associated with that key. Each function returns a type that corresponds to the function call name.

ecore_config_exit is then called to shutdown the Ecore_Config system before the program exits.

Example 4.2. Compilation command

```
gcc -o ecore_config_example ecore_config_example.c `ecore-config --cflags --libs`
```

To compile the program you can use the ecore-config script to get all of the required linking and library information for Ecore_Config. If you run this program as is you will receive the values put into ecore_config as the defaults as output. Once you know the program is working, you can create a simple config.db file to read the values.

Example 4.3. Simple config.db script (build_cfg_db.sh)

```
#!/bin/sh

DB=config.db

edb_ed $DB add /int_example int 2
edb_ed $DB add /this/is/a/string/example str "this is a string"
edb_ed $DB add /float/example float 42.10101
```

When build_cfg_db.sh is executed it will create a config.db file in the current directory. This file can then be copied into ~/.e/apps/PROGRAM_NAME/config.db where PROGRAM_NAME is the value passed into ecore_config_init. Once the file is copied in place, executing the test program again will show the values given in the config file instead of the defaults. You can specify as many, or as few of the configuration keys in the config file and Ecore_Config will either show the user value or the default value.

Chapter 5. EDB & EET

The libs.

Chapter 6. Esmart

Esmart provides a variety of EVAS smart objects that provide significant power to your EVAS and EFL based applications.

Recipe: Esmart Trans Introduction

dan sinclair <zero@perplexity.org>

Transparency is increasingly becoming a common trait of applications. To this end, the Esmart_Trans object has been created. This object will do all of the hard work to produce a transparent background for your program.

Esmart trans makes the integration of a transparent background into your application very easy. You need to create the trans object, and then make sure you update it as the window is moved or resized.

Example 6.1. Includes and declarations

```
#include <stdio.h>
#include <Ecore.h>
#include <Ecore_Evas.h>
#include <Esmart/Esmart_Trans.h>

int sig_exit_cb(void *data, int ev_type, void *ev);
void win_del_cb(Ecore_Evas *ee);
void win_resize_cb(Ecore_Evas *ee);
void win_move_cb(Ecore_Evas *ee);

static void _freshen_trans(Ecore_Evas *ee);
void make_gui();
```

Every application that uses an Esmart_Trans object is going to require the Ecore, Ecore_Evas and the Esmart/Esmart_Trans header files. The next four declarations are callbacks from ecore for events on our window, exit, delete, resize, and move respectively. The last two declarations are convenience functions being used in the example and do not need to be in your program.

Example 6.2. main

```
int main(int argc, char ** argv) {
    int ret = 0;

    if (!ecore_init()) {
        printf("Error initializing ecore\n");
        ret = 1;
        goto ECORE_SHUTDOWN;
    }

    if (!ecore_evas_init()) {
        printf("Error initializing ecore_evas\n");
        ret = 1;
        goto ECORE_SHUTDOWN;
    }
```

```
    make_gui();
    ecore_main_loop_begin();

    ecore_evas_shutdown();

ECORE_SHUTDOWN:
    ecore_shutdown();

    return ret;
}
```

The main routine for this example is pretty simple. Ecore and Ecore_Evas are both initialized, with appropriate error checking. We then create the gui and start the main ecore event loop. When ecore exits we shut everything down and return.

Example 6.3. exit and del callbacks

```
int sig_exit_cb(void *data, int ev_type, void *ev) {
    ecore_main_loop_quit();
    return 1;
}

void win_del_cb(Ecore_Evas *ee) {
    ecore_main_loop_quit();
}
```

The exit and del callbacks are the generic ecore callbacks.

Example 6.4. _freshen_trans

```
static void _freshen_trans(Ecore_Evas *ee) {
    int x, y, w, h;
    Evas_Object *o;

    if (!ee) return;

    ecore_evas_geometry_get(ee, &x, &y, &w, &h);
    o = evas_object_name_find(ecore_evas_get(ee), "bg");

    if (!o) {
        fprintf(stderr, "Trans object not found, bad, very bad\n");
        ecore_main_loop_quit();
    }
    esmart_trans_x11_freshen(o, x, y, w, h);
}
```

The _freshen_trans routine is a helper routine to update the image that the trans is shown. This will be called when we need to update our image to whats currently under the window. The function grabs the current size of the ecore_evas, and then gets the object with the name "bg" (this name is the same as the name we give our trans when we create it). Then, as long as the trans object exists, we tell esmart to

freshen the image being displayed.

Example 6.5. resize_cb

```
void win_resize_cb(Ecore_Evas *ee) {
    int w, h;
    int minw, minh;
    int maxw, maxh;
    Evas_Object *o = NULL;

    if (ee) {
        ecore_evas_geometry_get(ee, NULL, NULL, &w, &h);
        ecore_evas_size_min_get(ee, &minw, &minh);
        ecore_evas_size_max_get(ee, &maxw, &maxh);

        if ((w >= minw) && (h >= minh) && (h <= maxh) && (w <= maxw)) {
            if ((o = evas_object_name_find(ecore_evas_get(ee), "bg")))
                evas_object_resize(o, w, h);
        }
    }
    _freshen_trans(ee);
}
```

When the window is resized we need to update our evas to the correct size and then update the trans object to display that much of the background. We grab the current size of the window (ecore_evas_geometry_get) and the min/max size of the window. As long as our currently desired size is within the min/max bounds set for our window, we grab the "bg" (same as title again) object and resize it. Once the resizing is done, we call the _freshen_trans routine to update the image displayed on the bg.

Example 6.6. move_cb

```
void win_move_cb(Ecore_Evas *ee) {
    _freshen_trans(ee);
}
```

When the window is moved we need to freshen the image displayed as the transparency.

Example 6.7. Setup ecore/ecore_evas

```
void make_gui() {
    Evas *evas = NULL;
    Ecore_Evas *ee = NULL;
    Evas_Object *trans = NULL;
    int x, y, w, h;

    ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, sig_exit_cb, NULL);

    ee = ecore_evas_software_x11_new(NULL, 0, 0, 0, 300, 200);
    ecore_evas_title_set(ee, "trans demo");
```

```
    ecore_evas_callback_delete_request_set(ee, win_del_cb);
    ecore_evas_callback_resize_set(ee, win_resize_cb);
    ecore_evas_callback_move_set(ee, win_move_cb);

    evas = ecore_evas_get(ee);
```

The first portion of make_gui is concerned with setting up ecore and ecore_evas. First the exit callback is hooked into ECORE_EVENT_SIGNAL_EXIT, then the Ecore_Evas object is created with the software X11 engine. The window title is set and we hook in the callbacks written above, delete, resize and move. Finally we grab the evas for the created Ecore_Evas.

Example 6.8. Creating Esmart_Trans object

```
    trans = esmart_trans_x11_new(evas);
    evas_object_move(trans, 0, 0);
    evas_object_layer_set(trans, -5);
    evas_object_name_set(trans, "bg");

    ecore_evas_geometry_get(ee, &x, &y, &w, &h);
    evas_object_resize(trans, w, h);

    evas_object_show(trans);
    ecore_evas_show(ee);

    esmart_trans_x11_freshen(trans, x, y, w, h);
}
```

Once everything is setup we can create the trans object. The trans is to be created in the evas returned by ecore_evas_get. This initial creation is done by the call to esmart_trans_x11_new(evas). Once we have the object, we move it so it starts at position (0, 0) (the upper left corner), set the layer to -5 and name the object "bg" (as used above). Then we grab the current size of the ecore_evas and use that to resize the trans object to the window size. Once everything is resized we show the trans and show the ecore_evas. As a final step, we freshen the image on the transparency to what is currently under the window so it is up to date.

Example 6.9. Simple makefile

```
CFLAGS = `ecore-config --cflags` `evas-config --cflags` `esmart-config --cflags`
LIBS = `ecore-config --libs` `evas-config --libs` `esmart-config --libs`

all:
    gcc -o trans_example trans_example.c $(CFLAGS) $(LIBS)
```

In order to compile the above program we need to include the library information for ecore, ecore_evas and esmart. This is done through the -config scripts for each library. These -config scripts know where each of the includes and libraries resides and sets up the appropriate linking and include paths for the compilation.

Chapter 7. Etox & Estyle

Etox provides a power library for manipulation and layout of text in your EVAS/EFL application, which Estyle provides a veritile text stylization library and toolset that can be used with or without Etox.

Chapter 8. EWD

EWD is a library providing thread-safe basic data structures like hashes, lists and trees.

Chapter 9. Edje EDC

Edje Data Collections (EDC) source files allow for easy creation of rich and powerful interface designs...

Chapter 10. Edje

Edje is a complex graphical design and layout library.

Chapter 11. EWL

The Enlightenment Widget Library provides a full blown widget kit similar in feel to GTK, QT or MOTIF, but based apon the power of the Enlightenment Foundation Libraries.