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. Introducción

Bienvenido al estado iluminado de la programación. Este libro es una colección de ideas, trucos, tutoriales e introducciones dispuestas en estilo receta con objeto de ayudarte a convertirte rapidamente en compentente y efectivocon las Enlightenment Foundation Libraries. Las Enlightenment Foundation Libraries, llamadas simplemente EFL, son simplemente una colección de librerías originalmente escritas para soportar el gestor de ventanas Enlightenment DR17. Sin embargo, tal como las librerías crecieron y fueron sometidas a prueba e implantadas mayor y mas general funcionalidad fue añadida llevándonos a disfrutar un conjunto de librerías rico y potente que puede resolver todo tipo de problemas y tambien actuar como una venerable alternativa a las actualmente populares GTK y QT.

Las EFL son un grupo exhaustivo de librerías C que puden encargarse de casi cualquier necesidad gráfica en casi cualquier plataforma. La siguiente es una concisa descomposición de las librerías que componen EFL.

Lista de componentes EFL

Imlib2	Librería completa de manipulación de imagen, incluyendo renderizado a drawables X11.
EVAS	Librería de canvas con varios motores de backend incluyendo aceleración hardware OpenGL.
Ecore	Colección modular de librerías con manejo de eventos y temporizadores, mas sockets, IPC, inicialización, manejo de eventos, y finalización de FB y X, control de tareas, manejo de configuración, y más.
EDB	Una librería de base de datos capaz de almacenar cadenas, valores, y datos para manejo de configuración simple y centralizado.
EET	Un flexible formato de contenedor para almacenar imagenes binarias y datos.
Edje	Una librería y conjunto de herramientas de abstracción de imagen basado en EVAS, manteniendo todos los aspectos de interfaz de usuario completamente separados del codigo de la aplicación mediante el uso del paso de señales.
Embryo	Un lenguage de scripting tipicamente usado con Edje para control avanzado.
Etox	Una librería de formateado y manipulación de texto, completa con capacidades de estilización de texto.
Esmart	Una librería consistente de varios objetos inteligentes EVAS para facil reutilización, incluyendo el popular hack de transparencia.
Epeg	Una librería de thumbnailing veloz como el rayo para JPEG independiente de otros componentes EFL.
Epsilon	Una librería de thumbnailing flexible y rápida para PNG, XCF, GIF y JPEG. A flexable and fast thumbnailing library for PNG, XCF, GIF and JPEG.
Evoak	Una librería y conjunto de herrramientas de servidor canvas EVAS.
EWL	Una completa librería de widgets.
Emotion	Una librería de objetos EVAS para reproducción de video y DVD utilizando libxine.

Chapter 2. Imlib2

Imlib2 es la sucesora de Imlib. Noes simplemente una nueva versión - es una librería completamente nueva. Imlib2 puede ser instalada junto con Imlib 1.x dado que son efectivamente librerías distintas - pero tienen una funcionalidad similar.

Imlib2 puede hacer lo siguiente:

- Cargar imágenes de disco en uno de muchos formatos
- Grabar imágenes a disco en uno de muchos formatos
- Renderizar datos de imagen en otras imágenes
- Renderizar imágenes a un drawable X
- Producir pixmaps y mascaras de pixmap de imágenes
- Aplicar filtros a imágenes
- Rotar imágenes
- Aceptar datos RGBA para imágenes
- Scalar imágenes
- Hacer Alpha blend de Imágenes en otras imágenes o drawables
- Aplicar corrección de color y tablas de modificación y factores a imágenes
- Renderizar imágenes sobre imágenes con corrección de color y tablas de modificación
- Renderizar texto truetype con anti-aliasing
- Renderizar texto truetype con anti-aliasing a cualquier ángulo
- Renderizar líneas con anti-aliasing
- Renderizar rectángulos
- Renderizar gradientes lineales multi-coloreados
- Cachear datos intelligentemente para máximo rendimiento
- Obtener colores automaticamente
- Permitir control completo sobre el cacheado y obtención de color
- Proveer altamente optimizado ensamblador MMX para rutinas centrales
- Proveer interfaz para plug-in de filtros
- Proveer interfaz de carga y grabación de imágenes por plug-ins al vuelo en tiempo de ejecución
- La librería de composicionado, renderización y manipulación de imagen mas rapida para X

Si lo que quieres hacer no está en algún sitio en la lista de arriba, entonces probablemente Imlib2 no lo hace. Si lo hace, probáblemente lo hace mas rápido que cualquier otra librería que puedes encontrar (esto incluye gdk-pixbuf, gdkrgb, etc) principalmente a causa de código altamente optimizado y un subsistema inteligente que hace el trabajo sucio por ti y escoge las piezas por ti de manera que puedes ser perezoso y dejar que Imlib2 haga todas las optimizaciones por ti.

Imlib2 provee un potente motor para manipulación y renderizado de imagen. Usando cargadores puede manejar una variedad de formatos incluyendo BMP, GIF (via unGIF), JPEG, PNG, PNM, TGA, TIFF, XPM y otros.

Receta: marcado al agua para imágenes

Ben technikolor Rockwood <benr@cuddletech.com>

Con tanta gente poniendo tantas imágenes online es fácil olvidar de donde vinieron y dificil asegurar que material con copyright no es inadvertidamente mal utilizado. Simplemente añadiendo una imagen de marca de agua, como el logo de tu site, a cada una de tus imágenes puede resolver ambos problemas. Pero añadir marcas de agua manualmente es una tarea larga y repetitiva. Imlib2 puede fácilmente ser usada para resolver este problema. Lo que necesitamos hacer es tomar una imagen de entrada, y especificar una marca al agua (tu logo), posicionar la marca en la imagen de entrada, y entonces grabarla a una nueva imagen que usaremos en el site. La aplicación sería algo así:

Example 2.1. Imlib2 WaterMark Program

```
#define X_DISPLAY_MISSING
#include <Imlib2.h>
#include <stdio.h>

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

    Imlib_Image image_input, image_watermark, image_output;
    int     w_input, h_input;
    int     w_watermark, h_watermark;
    char    watermark[] = "watermark.png";

    if(argc > 1)  {
        printf("Input image is: %s\n", argv[1]);
        printf("Watermark is: %s\n", watermark);
    }
    else {
        printf("Usage: %s input_image output_imagename\n", argv[0]);
        exit(1);
    }

    image_input = imlib_load_image(argv[1]);
    if(image_input) {
        imlib_context_set_image(image_input);
        w_input = imlib_image_get_width();
        h_input = imlib_image_get_height();
        printf("Input size is: %d by %d\n", w_input, h_input);
        image_output = imlib_clone_image();
    }

    image_watermark = imlib_load_image(watermark);
    if(image_watermark) {
        imlib_context_set_image(image_watermark);
        w_watermark = imlib_image_get_width();
        h_watermark = imlib_image_get_height();
        printf("WaterMark size is: %d by %d\n",
                w_watermark, h_watermark);
    }

    if(image_output) {
        int dest_x, dest_y;

        dest_x = w_input - w_watermark;
        dest_y = h_input - h_watermark;
        imlib_context_set_image(image_output);

        imlib_blend_image_onto_image(image_watermark, 0,
                0, 0, w_watermark, h_watermark,
                dest_x, dest_y, w_watermark, h_watermark);
        imlib_save_image(argv[2]);
        printf("Wrote watermarked image to filename: %s\n", argv[2]);
    }


        return(0);
}
```

Mirando el ejemplo, primero hacemos una comprobación básica de argumentos, aceptando una imagen de entrada como primer argumento y un nombre de imagen de salida para nuestra copia marcada al agua. Usando `imlib_load_image()` cargamos la imagen de entrada y entonces obtenemos sus dimensiones usando las funciones get. Con `imlib_clone_image()` podemos crear una copia de la imagen de entrada, que será la base de nuestra imagen marcada al agua de salida. Despues cargamos la imagen de marca de agua, y observa que usamos `imlib_context_set_image()` para cambiar el contexto de la imágen de entrada a la imagen de marca de agua. Ahora obtenemos las dimensiones de la imagen tambien. En el bloque final hacemos dos cálculos simples para determinar el posicionamiento de la marca de agua en la imagen de output, en este caso quiero la marca de agua en la esquina inferior derecha. La función mágica que realmente hace el trabajo en este programa es `imlib_blend_image_onto_image()`. Observa que cambiamos el contexto a la imagen de salida antes de continuar. La función de blend, como su nombre indica (to blend = mezclar), mezcla dos imágenes juntas a las que referimos como fuente y destino. La función de mezcla mezcla una imagen fuente sobre la imagen en contexto actual, a la que designamos como destino. Los argumentos proporcionados a `imlib_blend_image_onto_image()` pueden parecer truculentos, necesitamos decirle que fuente utilizar (la marca de agua), si mezclar o on el canal alfa (0 para no), las dimensiones de la imagen fuente (x, y, w, h), y las dimensiones de la imagen destino (x, y, w, h). Notarás que en el ejemplo colocamos las posiciones x, y de la imagen de fuente (marca de agua) a 0 y usamos la anchura completa. El destino (imagen de entrada) se coloca en la esquina inferior derecha menos las dimensiones de la marca de agua, y entonces especificamos la anchura y altura de la marca de agua. Finalmente, usamos `imlib_save_image()` para grabar la imagen de salida.

Aunque este ejemplo debería ser significativamente mejorado para uso real, esboza la base del mezclado Imlib2 para resolver un problema muy común eficientemente.

Receta: Escalado de imagen

dan 'dj2' sinclair `<zero@perplexity.org>`

Conforme más gente obtiene la habilidad de poner imágenes en Internet es a menudo deseable escalar esas imágenes a un tamaño menor para reducir uso de anchura de banda. Esto puede ser fácilmente resuelto usando un simple programa Imlib2.

Esta receta toma el nombre de la imagen de entrada, la nueva anchura, altura y el nombre de la imagen de salida, y escala la imagen de entrada por los valores dados, grabándola en la imagen de salida.

Example 2.2. Escalado de imagen

```
#define X_DISPLAY_MISSING

#include <Imlib2.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char ** argv) {
    Imlib_Image in_img, out_img;
    int w, h;

    if (argc != 5) {
        fprintf(stderr, "Usage: %s [in_img] [w] [h] [out_img]\n", argv[0]);
        return 1;
    }

    in_img = imlib_load_image(argv[1]);
    if (!in_img) {
        fprintf(stderr, "Unable to load %s\n", argv[1]);
        return 1;
```

```
    }
    imlib_context_set_image(in_img);

    w = atoi(argv[2]);
    h = atoi(argv[3]);
    out_img = imlib_create_cropped_scaled_image(0, 0, imlib_image_get_width(),
                                                imlib_image_get_height(), w, h );
    if (!out_img) {
        fprintf(stderr, "Failed to create scaled image\n");
        return 1;
    }

    imlib_context_set_image(out_img);
    imlib_save_image(argv[4]);
    return 0;
}
```

Hay una mínima comprobaciñon de argumentos en este ejemplo, simplemente comprobando que tenemos el número correcto de argumentos.

La imagen fuente es cargada con una llamada a `imlib_load_image()`, la cual cargará los datos de imagen en memoria. Si la llamada falla, NULL será devuelto. En cuanto tengamos los datos de imagen necesitamos seleccionar la imagen para ser el contexto actual. Esto dice a Imlib2 en que imagen se efectuarán las operaciones. Esto se hace llamando `imlib_context_set_image()`. Cuando la imagen ha sido seleccionada como contexto actual podemos proceder con el escalado. Esto se hace llamando `imlib_create_cropped_scaled_image()`, que toma como argumentos, la posición de inicio en x, y , anchura y altura de la imagen fuente, y la anchura y altura de la imagen escalada. La razón por la que pasamos la información de la imagen fuente es que esta función tambien puede recortar la imagen si se desea. Para recortar, simplemente modifica x, y, anchura de fuente, y altura de fuente como sea deseado. Esto resultará en una nueva imagen out_img siendo producida. Si el escalado falla, NULL será devuelto. Entonces seleccionamos out_img como imagen de contexto actual y llamamos a la función de grabar, `imlib_save_image()`.

Aunque este programa es simple, enseña la simplicidad del escalado de imagen usando la API de Imlib2.

Receta: Rotación Libre

dan 'dj2' sinclair `<zero@perplexity.org>`

Es a veces deseable rotar una imagen en un ángulo específico. Imlib2 hace este proceso fácil. Este ejemplo muestra como es hecho. Si deseas rotar la imagen en ángulos de 90 grados o sus múltiplos, mira la recete de rotación de 90 grados dado que esta receta deja un borde negro alrededor de la imagen.

Example 2.3. Rotación libre

```
#define X_DISPLAY_MISSING

#include <Imlib2.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char ** argv) {
    Imlib_Image in_img, out_img;
    float angle = 0.0;
```

```
    if (argc != 4) {
        fprintf(stderr, "Usage: %s [in_img] [angle] [out_img]\n", argv[0]);
        return 1;
    }

    in_img = imlib_load_image(argv[1]);
    if (!in_img) {
        fprintf(stderr, "Unable to load %s\n", argv[1]);
        return 1;
    }
    imlib_context_set_image(in_img);

    angle = (atof(argv[2]) * (M_PI / 180.0));
    out_img = imlib_create_rotated_image(angle);
    if (!out_img) {
        fprintf(stderr, "Failed to create rotated image\n");
        return 1;
    }

    imlib_context_set_image(out_img);
    imlib_save_image(argv[3]);
    return 0;
}
```

Tras una mínima comprobación de argumentos llegamos al trabajo de Imlib2. Empezamos cargando la imagen especificada en memoria con `imlib_load_image()` dando el nombre de la imagen como parámetro. Entonces tomamos la imagen y la hacemos el contexto actual con `imblib_context_set_image`. Los contextos son usados en Imlib2 para decirle en que imagen trabajar. Cuando quieras hacer llamadas imlib en una imagen, debe estar seleccionada como contexto actual. Entonces convertimos el ángulo dado de grados a radianes dado que la función de rotación de Imlib2 trabaja en radianes. La rotación es hecha con `imlib_create_rotated_image()`. La función de rotación devolverá la nueva imagen. Para grabarla necesitamos seleccionarla como contexto actual, de nuevo con `imlib_context_set_image()`. Entonces una simple llamada a `imlib_save_image()` dando el nombre del archivo de salida graba la nueva imagen rotada.

La función de rotación en Imlib2 colocará un borde negro alrededor de la imagen para rellenar cualquier espacio vacío. Este borde es calculado de manera que la imagen rotada pueda caber en la salida. Esto colocará borders alrededor de la imagen incluso si la rotas por 180 grados.

Receta: Rotación de imagen de 90 grados

dan 'dj2' sinclair `<zero@perplexity.org>`

Con una cámara digital es a veces deseable rotar tu imagen 90, 180 o 270 grados. Esta receta enseñará como hacer esto fácilmente con Imlib2. Esta receta no pondrá bordes negros alrededor de la imagen como visto en el ejemplo de rotación libre.

Example 2.4. rotación de imagen de 90 grados

```
#define X_DISPLAY_MISSING

#include <Imlib2.h>
#include <stdlib.h>
#include <stdio.h>
```

```
int main(int argc, char ** argv) {
    Imlib_Image in_img;
    int dir = 0;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s [in_img] [out_img]\n", argv[0]);
        return 1;
    }

    in_img = imlib_load_image(argv[1]);
    if (!in_img) {
        fprintf(stderr, "Unable to load %s\n", argv[1]);
        return 1;
    }
    imlib_context_set_image(in_img);
    imlib_image_orientate(1);
    imlib_save_image(argv[2]);
    return 0;
}
```

Tras una mínima comprobación de errores cargamos la imagen a ser rotada con una llamada a `imlib_load_image()`. Esta función acepta un nombre de archivo y devuelve el objeto Imlib_Image, o NULL en caso de error de carga. Una vez la imagen ha sido cargada la seleccionamos como imagen de contexto actual, la imagen en la cual Imlib2 hará todas sus operaciones, con `imlib_context_set_image()`. La rotación es hecha mediante la llamada a `imlib_image_orientate()`. El parámetro para _orientate cambia la cantidad de rotación. Los valores posibles son: [1, 2, 3] siendo su significado rotación en el sentido de las agujas del reloj por [90, 180, 270] grados respectivamente. Una vez la imagen es rotada llamamos `imlib_save_image()` dando el nombre de archivo de la nueva imagen para que Imlib2 grabe la imagen rotada.

Con este ejemplo en tus manos deberías ser capaz de rotar imágenes rápidamente en intervalos de 90 grados usando Imlib2.

Receta: Reflexión de Imagen

dan 'dj2' sinclair `<zero@perplexity.org>`

Imlib2 contiene funciones para hacer reflexión de imagen. Esto puede ser hecho horizontal, vertical, o diagonalmente. Esta receta mostrará como implementar esta funcionalidad.

Example 2.5. Reflexión de Imagen

```
#define X_DISPLAY_MISSING

#include <Imlib2.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char ** argv) {
    Imlib_Image in_img;
    int dir = 0;

    if (argc != 4) {
        fprintf(stderr, "Usage: %s [in_img] [dir] [out_img]\n", argv[0]);
        return 1;
    }
```

```
    in_img = imlib_load_image(argv[1]);
    if (!in_img) {
        fprintf(stderr, "Unable to load %s\n", argv[1]);
        return 1;
    }
    imlib_context_set_image(in_img);

    dir = atoi(argv[2]);
    switch(dir) {
        case HORIZONTAL:
            imlib_image_flip_horizontal();
            break;

        case VERTICAL:
            imlib_image_flip_vertical();
            break;

        case DIAGONAL:
            imlib_image_flip_diagonal();
            break;

        default:
            fprintf(stderr, "Unknown value\n");
            return 1;
    }
    imlib_save_image(argv[3]);
    return 0;
}
```

Este ejemplo hace una mínima comprobación de argumentos, entonces carga la imagen de entrada usando `imlib_load_image()`, pasando el nombre de archivo a cargar. `imlib_load_image()` devolverá el objeto Imlib_Image, o NULL si la carga falla. En cuanto tenemos el objeto Image lo seleccionamos como contexto actual con una llamada a `imlib_context_set_image()`. Esto le dice a Imlib2 que esta es la imagen con la que queremos trabajar y todas las operaciones de Imlib2 funcionarán con esta imagen. Con el contexto de imagen configurado decidimos el tipo de reflexión que queremos efectuar. Esto se hace con una de las llamadas: `imlib_image_flip_horizontal()`, `imlib_image_flip_vertical()`, y `imlib_image_flip_diagonal()`. La rotación diagonal esencialmente coge la esquina superior izquierda y la hace la esquina inferior derecha, y viceversa. Una vez la imagen es reflejada llamamos `imlib_save_image()` dándole el nuevo nombre de archivo y ya hemos terminado.

Esto debería darte un ejemplo de reflexión de imagen con Imlib2. Necesitará mejoras antes de ser puesto en una aplicación real pero la base está ahí.

Chapter 3. EVAS

Evas es un API de canvas acelerado por hardware que puede trazar texto con anti-aliasing, imagenes suaves super y sub-sampleadas, hacer alpha-blending, asíi como limitarse a utilizar primitivas X11 normales como pixmaps, lineas, y rectangulos por velocidad si tu CPU o hardware gráfico son demasiado lentos.

Evas abstrae cualquier necesidad de conocer mucho sobre las características de la pantalla de tu servidor X, que profundidad o visuales mágicas tiene. Lo más que necesitas decir a Evas es cuantos colores (como máximo) utilizar si la pantalla no es una pantalla truecolor. Por defecto se sugiere usar 216 colores (dado que esto equivale a un cubo 6x6x6 ' exactamente el mismo cubo de color que Netscape, Mozilla, gdkrgb, etc utilizan de manera que los colores serán compartidos). Si Evas no puede obtener suficientes colores continúa reduciendo el tamaño del cubo de color hasta que alcanza blanco y negro. De esta manera, puede dibujar en cualquier cosa desde un terminal blanco y negro, VGA 16 colores, 256 colores y todo el camino a color 15, 16, 24y 32bit.

Receta: Usando Ecore_Evas para simplificar inicialización de canvas X11

Ben technikolor Rockwood

Evas es una librer'ia potente y simple de usar, pero antes de establecer un canvas un drawable X11 tiene que ser configurado. Configurar manualmente X11 puede ser una tarea frustrante y caótica que te impide concentrarte en lo que realmente quieres hacer: desarrolar una aplicación Evas. Pero todo esto puede ser evitado usando el modulo Ecore_Evas de ecore para hacer todo el trabajo sucio por ti.

El siguiente ejemplo es una plantilla básica que puede ser usada como punto de partida para cualquier aplicación Evas que quieras construir, significativamente reduciendo tiempo de desarrollo.

Example 3.1. Plantilla Ecore_Evas

```
#include <Ecore_Evas.h>
#include <Ecore.h>

#define WIDTH 400
#define HEIGHT 400

        Ecore_Evas  *   ee;
        Evas        *   evas;
        Evas_Object *   base_rect;

int main(){

        ecore_init();

   ee = ecore_evas_software_x11_new(NULL, 0,  0, 0, WIDTH, HEIGHT);
        ecore_evas_title_set(ee, "Ecore_Evas Template");
        ecore_evas_borderless_set(ee, 0);
        ecore_evas_show(ee);


   evas = ecore_evas_get(ee);
        evas_font_path_append(evas, "fonts/");
```

```
   base_rect = evas_object_rectangle_add(evas);
        evas_object_resize(base_rect, (double)WIDTH, (double)HEIGHT);
        evas_object_color_set(base_rect, 244, 243, 242, 255);
        evas_object_show(base_rect);

        /* Insert Object Here */

        ecore_main_loop_begin();

        return 0;
}
```

Detelles completos sobre Ecore_Evas pueden encontrarse en el capítulo de Ecore en este libro, pero esta Plantilla básica debería permitirte jugar con Evas inmediatamente. Las llamadas importantes que remarcar son `ecore_evas_borderless_set()` que define si la ventana Evas está enmarcada por tu gestor de ventanas o sin borde , y `evas_font_path_append()` que define el/los path(s) de fuentes utilizados por tu aplicación Evas.

Receta: Vínculos de teclado, usando eventos de teclado EVAS

Ben technikolor Rockwood

Muchas aplicaciones proveen vínculos de teclado para operaciones comunmente usadas. Bien aceptando texto en maneras que las EFL no esperan, o simplemente una manera de vincular la tecla + a subir el volumen de un mezclador, los vínculos de teclado pueden añadir la pequeña funcionalidad que hace tu aplicación un éxito.

El siguiente código es una simple y completa aplicación útil para explorar vínculos de teclado usando callbacks de evento EVAS. Crea una ventana negra de 100 por 100 píxels en la cual puedes presionar teclas.

Example 3.2. Captura de teclado usando eventos EVAS

```
#include <Ecore_Evas.h>
#include <Ecore.h>

#define WIDTH 100
#define HEIGHT 100

        Ecore_Evas  *   ee;
        Evas        *   evas;
        Evas_Object *   base_rect;

static int main_signal_exit(void *data, int ev_type, void *ev)
{
   ecore_main_loop_quit();
   return 1;
}

void key_down(void *data, Evas *e, Evas_Object *obj, void *event_info) {
        Evas_Event_Key_Down *ev;

        ev = (Evas_Event_Key_Down *)event_info;
        printf("You hit key: %s\n", ev->keyname);
```

```
}

int main(){
        ecore_init();
        ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT,
                        main_signal_exit, NULL);

   ee = ecore_evas_software_x11_new(NULL, 0,  0, 0, WIDTH, HEIGHT);
        ecore_evas_title_set(ee, "EVAS KeyBind Example");
        ecore_evas_borderless_set(ee, 0);
        ecore_evas_show(ee);

   evas = ecore_evas_get(ee);

   base_rect = evas_object_rectangle_add(evas);
        evas_object_resize(base_rect, (double)WIDTH, (double)HEIGHT);
        evas_object_color_set(base_rect, 0, 0, 0, 255);
        evas_object_focus_set(base_rect, 1);
        evas_object_show(base_rect);

        evas_object_event_callback_add(base_rect,
                        EVAS_CALLBACK_KEY_DOWN, key_down, NULL);

        ecore_main_loop_begin();

        ecore_evas_shutdown();
        ecore_shutdown();

        return 0;
}
```

Puedes compilar este ejemplo de la siguiente manera:

Example 3.3. Compilación de vínvulos de teclado EVAS

```
gcc `evas-config --libs --cflags` `ecore-config --libs --cflags` \
> key_test.c -o key_test
```

En este ejemplo el canvas es iniciado de la manera habitual usando Ecore_Evas para hacer el trabajo sucio. La magia ocurre en `evas_object_event_callback_add()`

```
        evas_object_event_callback_add(base_rect,
                        EVAS_CALLBACK_KEY_DOWN, key_down, NULL);
```

Añadiendo un callback a base_rect, que actua como background del canvas, podemos ejecutar una función (`key_down()` en este caso) cuando encontremos un evento de tecla pulsada definido en `Evas.h` como EVAS_CALLBACK_KEY_DOWN.

Hay una cosa muy importante que hacer además de definir el callback: capturar el foco. La función `evas_object_focus_set()` captura el foco en un objeto Evas dado. Es el objeto que tiene el foco el que aceptará los eventos, aún cuando tu explicitamente defines el objeto Evas al cual el callback está vinculado. Y solo un objeto puede tener el foco a la vez. El problema mas común encontrado con los callbacks Evas es olvidar capturar el foco.

```
void key_down(void *data, Evas *e, Evas_Object *obj, void *event_info) {
        Evas_Event_Key_Down *ev;

        ev = (Evas_Event_Key_Down *)event_info;
        printf("You hit key: %s\n", ev->keyname);
}
```

La función `key_down()` es llamada cuando un evento de tecla pulsada ocurre despues de definir su callback. La declaración de la función es la de un callback estándar Evas (ver `Evas.h`). La pieza importante de información que necesitamos saber es que tecla fue pulsada, que está contenida en la structura Evas `event_info`. Tras iniciar la estructura `Evas_Event_Key_Down` como visto arriba podemos acceder el elemento `keyname` para determinar que tecla fue pulsada.

En la mayoria de los casos probablemente usarás un `switch` o `if`s anidados para definir que teclas hacen que cosa, y se recomienda que esta funcioonalidad sea complementada con una configuración EDB para proveer centralización y facil expansion de la configuración de vínculos de teclado de tus aplicaciones.

Receta: Introducción a los objetos inteligentes Evas

dan 'dj2' sinclair <zero@perplexity.org>

Cuando trabajes mas con Evas, empezarás a tener varios `Evas_Object` con los cuales estás trabajando y aplicando las mismas operaciones para mantenerlos en sincronía. Sería mucho mas conveniente agrupar todos esos `Evas_Object` en un único objeto al que todas las transformaciones pueden ser aplicadas.

Los objetos inteligentes Evas proveen la capacidad de escribir tus propios objetos y tener a Evas llamando a tus funciones para hacer el movimiento, redimensionado, ocultamiento, capas y todas las cosas que un `Evas_Object` es responsable de manejar. Junto con los callback normales de `Evas_Object`, los objetos inteligentes te permiten definir tus propias funciones para manejar operaciones especiales que puedas requerir.

Esta introducción está partida en 3 archivos: foo.h, foo.c y main.c. El objeto inteligente creado se llama `foo` y es definido en los archivos foo.[ch]. maic.c está para mostrar como el nuevo objeto inteligente puede ser usado.

El objeto inteligente es solamente dos cuadrados, uno dentro del otro, con el interno separado un 10% del borde del externo. Conforme el programa se ejecute un callback de temporizador Ecore reposicionará y actualizará el tamaño del objecto inteligente.

El archivo básico para este objeto inteligente es de una plantilla Evas Smart Object por Atmos localizada en: `www.atmos.org/code/src/evas_smart_template_atmos.c [http://www.atmos.org/code/src/evas_smart_template_atmos.c]` que a su vez estaba basada en una plantilla por Rephorm.

Primero necesitamos definir el interfaz externo a nuestro objeto inteligente. En este caso solo necesitamos una llamada para crear el nuevo objeto.

Example 3.4. foo.h

```
#ifndef _FOO_H_
```

```
#define _FOO_H_

#include <Evas.h>

Evas_Object *foo_new(Evas *e);

#endif
```

Con eso fuera del camino, nos metemos en las oscuras entra;as de la bestia, el código del objeto inteligente.

Example 3.5. foo.c

```
#include <Evas.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct _Foo_Object Foo_Object;
struct _Foo_Object {
        Evas_Object *clip;
        Evas_Coord x, y, w, h;

        Evas_Object *outer;
        Evas_Object *inner;
};
```

`Foo_Object` almacenará toda la información que necesitamos para nuestro objeto. En este caso es el objecto caja externa, el objeto caja interna, un objeto de clipping y la actual posición y tamaño del objeto.

```
static Evas_Smart *_foo_object_smart_get();
static Evas_Object *foo_object_new(Evas *evas);
static void _foo_object_add(Evas_Object *o);
static void _foo_object_del(Evas_Object *o);
static void _foo_object_layer_set(Evas_Object *o, int l);
static void _foo_object_raise(Evas_Object *o);
static void _foo_object_lower(Evas_Object *o);
static void _foo_object_stack_above(Evas_Object *o, Evas_Object *above);
static void _foo_object_stack_below(Evas_Object *o, Evas_Object *below);
static void _foo_object_move(Evas_Object *o, Evas_Coord x, Evas_Coord y);
static void _foo_object_resize(Evas_Object *o, Evas_Coord w, Evas_Coord h);
static void _foo_object_show(Evas_Object *o);
static void _foo_object_hide(Evas_Object *o);
static void _foo_object_color_set(Evas_Object *o, int r, int g, int b, int a);
static void _foo_object_clip_set(Evas_Object *o, Evas_Object *clip);
static void _foo_object_clip_unset(Evas_Object *o);
```

Las predeclaraciones requeridas para el objeto inteligente. Estas serán explicadas conforme lleguemos a la implementación.

```
Evas_Object *foo_new(Evas *e) {
    Evas_Object *result = NULL;
```

```
    Foo_Object *data = NULL;

    if ((result = foo_object_new(e))) {
        if ((data = evas_object_smart_data_get(result)))
            return result;
        else
            evas_object_del(result);
    }

    return NULL;
}
```

`foo_new()` es nuestro único interfaz externo y es responsable de configurar el propio objeto. La llamada a `foo_object_new()` hará el trabajo pesado en la creación del objeto. La `evas_object_smart_data_get()` es mas una comprobación de error que otra cosa. Cuando `foo_object_new()` se ejecuta añadirá el objeto inteligente al evas y esto resultará en una llamada `add` creará un `Foo_Object`. Así , solamente estamos comprobando que `Foo_Object` ha sido creado.

```
static Evas_Object *foo_object_new(Evas *evas) {
    Evas_Object *foo_object;

    foo_object = evas_object_smart_add(evas, _foo_object_smart_get());
    return foo_object;
}
```

Nuestra función `foo_object_new()` tiene la simple tarea de añadir nuestro objeto inteligente en un Evas dado. Esto se hace a través de `evas_object_smart_add()` pasando el Evas y el objeto `Evas_Smart *`. Nuestro `Evas_Smart *` es producido por la llamada `_foo_object_smart_get()`.

```
static Evas_Smart *_foo_object_smart_get() {
    static Evas_Smart *smart = NULL;
    if (smart)
        return (smart);

    smart = evas_smart_new("foo_object",
                           _foo_object_add,
                           _foo_object_del,
                           _foo_object_layer_set,
                           _foo_object_raise,
                           _foo_object_lower,
                           _foo_object_stack_above,
                           _foo_object_stack_below,
                           _foo_object_move,
                           _foo_object_resize,
                           _foo_object_show,
                           _foo_object_hide,
                           _foo_object_color_set,
                           _foo_object_clip_set,
                           _foo_object_clip_unset,
                           NULL
                          );
    return smart;
}
```

Notarás que `Evas_Smart *smart` en esta función es declarado `static`.Esto es porque no importa cuantos `Evas_Object` creemos , sólo habrá un objeto `Evas_Smart`. Tal como Raster lo pone, `Evas_Smart` es como una definición de clase C++, no una instancia. El `Evas_Object` es la instan-

cia del `Evas_Smart`.

El objeto inteligente en sí es creado a través de la llamada a la función `evas_smart_new()`. A esta función pasamos el nombre del objeto inteligente, todas las rutinas callback para el objeto inteligente, y cualesquiera datos de usuario. En este caso no tenemos datos de usuario así que usamos NULL.

```
static void _foo_object_add(Evas_Object *o) {
    Foo_Object *data = NULL;
    Evas *evas = NULL;

    evas = evas_object_evas_get(o);

    data = (Foo_Object *)malloc(sizeof(Foo_Object));
    memset(data, 0, sizeof(Foo_Object));

    data->clip = evas_object_rectangle_add(evas);
    data->outer = evas_object_rectangle_add(evas);
    evas_object_color_set(data->outer, 0, 0, 0, 255);
    evas_object_clip_set(data->outer, data->clip);
    evas_object_show(data->outer);

    data->inner = evas_object_rectangle_add(evas);
    evas_object_color_set(data->inner, 255, 255, 255, 126);
    evas_object_clip_set(data->inner, data->clip);
    evas_object_show(data->inner);

    data->x = 0;
    data->y = 0;
    data->w = 0;
    data->h = 0;

    evas_object_smart_data_set(o, data);
}
```

Cuando `evas_object_smart_add()` es llamada en `foo_object_new()`, esta función `_foo_object_add()` será llamada de manera que podemos configurar cualesquiera datos internos para este objeto inteligente.

Para este objeto inteligente tres `Evas_Object` internos. Siendo estos `data->clip`, usado para hacer clipping con los otros dos objectos, `data->outer`, nuestro rectángulo externo y `data->inner`, nuestro rectángulo interno. Los rectángulos interno y externo tienen el clipping puesto en el objeto clip y son mostrados inmediatamente. El objeto clip no es mostrado, lo será cuando el usuario llame `evas_object_show()` en este objecto.

Finalmente llamamos `evas_object_smart_data_set()` para colocar nuestro nuevo `Foo_Object` como datos para este objecto inteligente. Estos datos serán obtenidos en otras funciones de este objeto llamando a `evas_object_smart_data_get()`.

```
static void _foo_object_del(Evas_Object *o) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        evas_object_del(data->clip);
        evas_object_del(data->outer);
        evas_object_del(data->inner);
        free(data);
    }
}
```

El callback `_foo_object_del()` será ejecutado si el usuario llama `evas_object_del()` en nuestro objeto. Para este objecto es tan simple como llamar a `evas_object_del` en nuestros 3 rectangulos y liberar la estructura `Foo_Object`.

```
static void _foo_object_layer_set(Evas_Object *o, int l) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        evas_object_layer_set(data->clip, l);
    }
}

static void _foo_object_raise(Evas_Object *o) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        evas_object_raise(data->clip);
    }
}

static void _foo_object_lower(Evas_Object *o) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        evas_object_lower(data->clip);
    }
}

static void _foo_object_stack_above(Evas_Object *o, Evas_Object *above) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        evas_object_stack_above(data->clip, above);
    }
}

static void _foo_object_stack_below(Evas_Object *o, Evas_Object *below) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        evas_object_stack_below(data->clip, below);
    }
}
```

Este grupo de funciones: `_foo_object_layer_set()`, `_foo_object_raise()`, `_foo_object_lower()`, `_foo_object_stack_above()`, y `_foo_object_stack_below()` funcionan todos de la misma manera, aplicando la función requerida `evas_object_*` al objeto `data->clip`.

Estas funciones son activadas por el uso de: `evas_object_layer_set()`, `evas_object_raise()`, `evas_object_lower()`, `evas_object_stack_above()`, y `evas_object_stack_below()` respectivamente.

```
static void _foo_object_move(Evas_Object *o, Evas_Coord x, Evas_Coord y) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        float ix, iy;
        ix = (data->w - (data->w * 0.8)) / 2;
        iy = (data->h - (data->h * 0.8)) / 2;
```

```
        evas_object_move(data->clip, x, y);
        evas_object_move(data->outer, x, y);
        evas_object_move(data->inner, x + ix, y + iy);

        data->x = x;
        data->y = y;
    }
}
```

El callback `_foo_object_move()` será accionado cuando `evas_object_move()` sea llamado en nuestro objecto. Cada uno de los objetos internos es movido a su posición correcta con llamadas a `evas_object_move()`.

```
static void _foo_object_resize(Evas_Object *o, Evas_Coord w, Evas_Coord h) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o))) {
        float ix, iy, iw, ih;
        iw = w * 0.8;
        ih = h * 0.8;

        ix = (w - iw) / 2;
        iy = (h - iw) / 2;

        evas_object_resize(data->clip, w, h);
        evas_object_resize(data->outer, w, h);

        evas_object_move(data->inner, data->x + ix, data->y + iy);
        evas_object_resize(data->inner, iw, ih);

        data->w = w;
        data->h = h;
    }
}
```

El callback `_foo_object_resize()` será accionado cuando el usuario llame `evas_object_resize()` en nuestro objeto. Así, para nuestro objecto, necesitamos redimensionar `data->clip` y `data->outer` al tamaño completo disponible para nuestro objecto. Esto es hecho con las llamadas a `evas_object_resize()`. Entonces necesitamos mover y redimensionar el objeto `data->inner` de manera que permanezca en la posición correcta en el rectángulo externo. Esto es hecho con `evas_object_move()` y `evas_object_resize()` respectivamente. Entonces almacenamos de nuevo la anchura y altura en nuestro objeto de manera que podemos referenciarlas mas tarde.

```
static void _foo_object_show(Evas_Object *o) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o)))
            evas_object_show(data->clip);
}
```

El callback `_foo_object_show()` será accionado cuando `evas_object_show()` sea llamada en nuestro objecto. Para mostrar nuestro objeto todo lo que necesitamos es mostrar la región de clip, dado que nuestros rectángulos están en ella. Esto se hace en la llamada a `evas_object_show()`.

```
static void _foo_object_hide(Evas_Object *o) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o)))
        evas_object_hide(data->clip);
}
```

El callback `_foo_object_hide()` será accionado cuando una llamada a `evas_object_hide()` sea hecha en nuestro objecto. Como estamos usando un objeto de clipping interno solo necesitamos esconder el objecto clip, `data->clip`, para esconder nuestro objecto inteligente. Esto se hace por medio de la llamada a `evas_object_hide()`.

```
static void _foo_object_color_set(Evas_Object *o, int r, int g, int b, int a) {
}
```

La función `_foo_object_color_set()` será llamada cuando `evas_object_color_set()` sea llamada en nuestro `Evas_Object`. Pero, dado que no quiero que mi objeto cambie de colores, ignoro este callback.

```
static void _foo_object_clip_set(Evas_Object *o, Evas_Object *clip) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o)))
        evas_object_clip_set(data->clip, clip);
}
```

El callback `_foo_object_clip_set()` será accionado cuando la llamada `evas_object_clip_set()` sea hecha en nuestro objeto. En este caso propagamos esto a nuestro objeto de clipping interno por medio de la llamada a `evas_object_clip_set()`.

```
static void _foo_object_clip_unset(Evas_Object *o) {
    Foo_Object *data;

    if ((data = evas_object_smart_data_get(o)))
        evas_object_clip_unset(data->clip);
}
```

El callback `_foo_object_clip_unset()` será accionado cuando una llamada a `evas_object_clip_unset()` sea hecha en nuestro objecto. Simplemente borramos el clip interno por medio de la llamada a `evas_object_clip_unset()`.

Una vez el código de objeto inteligente está completo podemos crear nuestro programa principal para utilizar el nuevo objeto inteligente.

Example 3.6. main.c

```
#include <stdio.h>
#include <Ecore_Evas.h>
#include <Ecore.h>
#include "foo.h"

#define WIDTH 400
#define HEIGHT 400
```

```
#define STEP 10

static int dir = -1;
static int cur_width = WIDTH;
static int cur_height = HEIGHT;

static int timer_cb(void *data);

int main() {
    Ecore_Evas  *ee;
    Evas        *evas;
    Evas_Object *o;

    ecore_init();

    ee = ecore_evas_software_x11_new(NULL, 0, 0, 0, WIDTH, HEIGHT);
    ecore_evas_title_set(ee, "Smart Object Example");
    ecore_evas_borderless_set(ee, 0);
    ecore_evas_show(ee);

    evas = ecore_evas_get(ee);

    o = evas_object_rectangle_add(evas);
    evas_object_resize(o, (double)WIDTH, (double)HEIGHT);
    evas_object_color_set(o, 200, 200, 200, 255);
    evas_object_layer_set(o, -255);
    evas_object_show(o);

    o = foo_new(evas);
    evas_object_move(o, 0, 0);
    evas_object_resize(o, (double)WIDTH, (double)HEIGHT);
    evas_object_layer_set(o, 0);
    evas_object_show(o);

    ecore_timer_add(0.1, timer_cb, o);
    ecore_main_loop_begin();

    return 0;
}

static int timer_cb(void *data) {
    Evas_Object *o = (Evas_Object *)data;
    Evas_Coord x, y;

    cur_width += (dir * STEP);
    cur_height += (dir * STEP);

    x = (WIDTH - cur_width) / 2;
    y = (HEIGHT - cur_height) / 2;

    if ((cur_width < STEP) || (cur_width > (WIDTH - STEP)))
        dir *= -1;

    evas_object_move(o, x, y);
    evas_object_resize(o, cur_width, cur_height);
    return 1;
}
```

La mayor parte de este programa es el mismo que el dado en la receta dada previamente usando Ecore_Evas. La creación de nuestro nuevo objeto inteligente es vista en el fragmento de código:

```
    o = foo_new(evas);
```

```
    evas_object_move(o, 0, 0);
    evas_object_resize(o, (double)WIDTH, (double)HEIGHT);
    evas_object_layer_set(o, 0);
    evas_object_show(o);
```

Una vez tu nueva `foo_new()` devuelve nuestro objeto podemos manipularlo con las llamadas Evas normales, así que procedemos a colocar su posición , tamaño, capa y entonces a mostrar el objeto.

Una vez el nuvo objeto inteligente es creado y mostrado configuramos un temporizador Ecore para accionarse cada 0.1 segundos. Cuando el evento sea accionado ejecutará la función `timer_cb()`. Esta función callback disminuirá o aumentará el tamaño de nuestro objeto inteligente a la vez que lo mantiene centrado en la ventana principal.

Compilar este ejemplo es tan simple como:

Example 3.7. Compilación

```
zero@oberon [evas_smart] -> gcc -o foo foo.c main.c \
    `ecore-config --cflags --libs` `evas-config --cflags --libs`
```

Los objetos inteligentes Esmart son simples de crear pero proveen un potente mecanismo para abstraer piezas de tu programa. Para ver mas objetos inteligentes echa un vistazo a cualquiera de los objetos Esmart, Etox o Emotion.

Chapter 4. Ecore

Â¿Que es ecore ? Ecore es la capa central de abstracción de eventos y abstracción X que permite hacer selecciones, Xdnd, cosas de X en general, y cilos de evento, eventos de tiempo desocupado y tiempo expirado, de una manera rápida, optimizada y conveniente. Es una librería separada de manera que cualquiera puede hacer uso del trabajo puesto en Ecore para hacer este trabajo fácil para aplicaciones.

Ecore es completamente modular. En su base están los temporizadores y manejadores de eventos, y funciones de inicialización y cierre. Los módulos de abstracción para Ecore incluyen:

- Ecore X
- Ecore FB
- Ecore EVAS
- Ecore TXT
- Ecore Job
- Ecore IPC
- Ecore Con
- Ecore Config

Ecore es tan modular y potente que puede ser muy util incluso en programación no gráfica. Como ejemplo, varios servidores web han sido escritos basados solo en Ecore y el módulo Ecore_Con para abstraer comunicación pro socket.

Receta: Introducción a la configuración de Ecore

dan 'dj2' sinclair <zero@perplexity.org>

El modulo Ecore_Config provee al programadorcon una manera muy simple de preparar archivos de configuración para su programa. Esta receta dará un ejemplo de como integrar los principios de Ecore_Config en tu programa y utilizarla para conseguir datos de configuración.

Example 4.1. Programa simple Ecore_Config

```
#include <Ecore_Config.h>

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

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

    ecore_config_int_default("/int_example", 1);
    ecore_config_string_default("/this/is/a/string/example", "String");
    ecore_config_float_default("/float/example", 2.22);

    ecore_config_load();

    i = ecore_config_int_get("/int_example");
    str = ecore_config_string_get("/this/is/a/string/example");
```

```
    j = ecore_config_float_get("/float/example");

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

    free(str);

    ecore_config_shutdown();
    return 0;
}
```

Como puedes ver el uso básico de Ecore?Config es simple. El sistema es inicializado con una llamada a `ecore_config_init`. El nombre del programa controla donde Ecore_Config mirará por tu base de datos de configuración. El directorio y nombre de archivo son: ~/.e/apps/NOMBRE_DEL_PROGRAMA/config.db.

Para cada variable de configuración que recibes de Ecore_Config, puedes asignar un valor por defecto por si acaso el usuario no tiene un archivo config.db. Los valores por defecto son asignados con ecore_config_*_default donde * es uno de los tipos Ecore_Config. El primer parámetro es la clave bajo la cual va a ser accedida. Estas claves deben de ser únicas en tu programa. El valor pasado es del tipo apropiado para esta llamada.

La llamada a `ecore_config_load` leerá los valores del archivo config.db en Ecore_Config. Despues de lo cual podemos acceder los ficheros con los métodos ecore_config_*_get (de nuevo * es el tipo de datos deseado). Estas rutinas toman el nombre de clave para este item y retornan el valor asociado con esa clave. Cada función retorna un tipo que corresponde al nombre de llamada a la función. call will read the values from the config.db file into Ecore_Config.

`ecore_config_shutdown` es llamada entonces para terminar el sistema Ecore_Config antes de salir del programa.

Example 4.2. Comando de compilación

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

Para compilar el programa puedes usar la script ecore-config para conseguir toda la información de librería y enlazado requerida para Ecore_Config. Si ejecutas este programa tal cual recibirás como output los valores puestos en ecore_config. Una vez sabes que el programa está trabajando, puedes crear un archivo config.db simple para leer los valores.

Example 4.3. Script simple config.db (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
```

Cuando build_cfg_db.sh sea ejecutada creará un archivo config.db en el directorio actual. Este archivo puede entonces ser copiado en ~/.e/apps/NOMBRE_DEL_PROGRAMA/config.db donde NOMBRE_DEL_PROGRAMA es el valor pasado en ecore_config_init. Una vez que el archivo sea copiado en su lugar, ejecutar el programa de nuevo mostrará los valores dados en el archivo config en lugar de los valores por defecto. Puedes especificar tantas o tan pocas de las claves de configuración en el archivo config y Ecore_Config mostrará el valor de usuario o el valor por defecto.

Receta: Oyentes Ecore Config

dan 'dj2' sinclair <zero@perplexity.org>

Cuando se usa Ecore Config para manejar la configuración de tu aplicación esbueno ser notificado cuando esa configuración ha sido cambiada. Esto se consigue mediante el uso de oyentes en Ecore_Config.

Example 4.4. Oyente Ecore_Config

```
#include <Ecore.h>
#include <Ecore_Config.h>

static int listener_cb(const char *key, const Ecore_Config_Type type,
            const int tag, void *data);

enum {
    EX_ITEM,
    EX_STR_ITEM,
    EX_FLOAT_ITEM
};

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

    if (!ecore_init()) {
        printf("Cannot init ecore");
        return 1;
    }

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

    ecore_config_int_default("/int/example", 1);
    ecore_config_string_default("/string/example", "String");
    ecore_config_float_default("/float/example", 2.22);

    ecore_config_listen("int_ex", "/int/example", listener_cb,
                                                  EX_ITEM, NULL);
    ecore_config_listen("str_ex", "/string/example", listener_cb,
                                                  EX_STR_ITEM, NULL);
    ecore_config_listen("float_ex", "/float/example", listener_cb,
                                                  EX_FLOAT_ITEM, NULL);

    ecore_main_loop_begin();
    ecore_config_shutdown();
```

```
    ecore_shutdown();
    return 0;
}

static int listener_cb(const char *key, const Ecore_Config_Type type,
                                              const int tag, void *data) {

    switch(tag) {
        case EX_ITEM:
            {
                int i = ecore_config_int_get(key);
                printf("int_example :: %d\n", %i);
            }
            break;

        case EX_STR_ITEM:
            {
                char *str = ecore_config_string_get(key);
                printf("str :: %s\n", %str);
                free(str);
            }
            break;

        case EX_FLOAT_ITEM:
            {
                float f = ecore_config_float_get(key);
                printf("float :: %f\n", %f);
            }
            break;

        default:
            printf("Unknown tag (%d)\n", tag);
            break;
    }
}
```

Ecore_Config es iniciado de la manera habitual, y creamos algunas claves por defecto como normalmente ocurre. Las partes interesantes entran en juego con las llamadas a `ecore_config_listen()`. Esta es la llamada que le dice a Ecore_Config que queremos ser notificados de cambios en la configuración. `ecore_config_listen()` toma cinco parametros:

- name
- key
- listener callback
- id tag
- user data

El campo `name`es una cadena de nombre dado por ti para identificar este callback de oyente. `key` es el nombre de la clave en la cual quieres escuchar, este será el mismo que el nombre de clave dado en las llamadas _default arriba. `listener callback` es la función callback a ser ejecutada en caso de cambio. `id tag` es una etiqueta numérica que puede ser dada a cada oyente y será pasada a la función callback. Finalmente, `user data` son los dats que quieras que sean pasados al callback cuando se ejecute.

La función callback tiene una firma semejante a:

```
int listener_cb(const char *key, const Ecore_Config_Type type,
                                       const int tag, void *data);
```

`key` es el nombre de clave en el que escuchar. El parámetro `type` contendrá el tipo Ecore_Config. Este

puede ser uno de:

PT_NIL	Propiedad sin valor
PT_INT	Propiedad Integer
PT_FLT	Propiedad Float
PT_STR	Propiedad String
PT_RGB	Propiedad Colour
PT_THM	Propiedad Theme

El parámetro `tag` es el valor por defecto que fue dado en la llamada de creación de oyente mas arriba. Finalmente, `data` son cualesquiera datos adjuntados al oyente cuando fue creado.

Si quieres eliminar el oyente en un momento posterior usa `ecore_config_deaf()`. Esta toma tres parámetros:

- name
- key
- listener callback

Cada uno de estos parámetros corresponde al parámetro dado en la llamada inicial a `ecore_config_listen()`.

Example 4.5. Compilación

```
zero@oberon [ecore_config] -> gcc -o ecfg ecfg_listener.c \
     `ecore-config --cflags --libs`
```

Si ejecutas el programa verás los valores por defecto escritos en la pantalla. Si lanzas ahora examine como sigue:

```
zero@oberon [ecore_config] -> examine foo
```

(foo es el nombre dado a `ecore_config_init()`). Deberías entonces ser capaz de modificar la configuración de la aplicación y, tras pulsar 'save', ver en la consola los valores modificados.

Receta: Introducción a Ecore Ipc

dan 'dj2' sinclair <zero@perplexity.org>

La librería Ecore_Ipc provee un envoltorio robusto y eficiente alrededor del modulo Ecore_Con. Ecore_Ipc te permite iniciar las comunicaciones de tu servidor y maneja todos los asuntos truculentos internos. Esta receta dará un ejemplo simple de un cliente Ecore y un servidor Ecore.

Cuando trabajamos con Ecore_Ipc, escribiendo un cliente o un servidor un objeto Ecore_Ipc_Server será creado. Esto es porque en cualquier caso hay un servidor siendo manipulado, bien el que se está iniciando, o con el que nos comunicamos. Despues de eso, todo es facil.

Example 4.6. Cliente Ecore_Ipc: preámbulo

```
#include <Ecore.h>
#include <Ecore_Ipc.h>

int sig_exit_cb(void *data, int ev_type, void *ev);
int handler_server_add(void *data, int ev_type, void *ev);
int handler_server_del(void *data, int ev_type, void *ev);
int handler_server_data(void *data, int ev_type, void *ev);
```

El archivo Ecore.h es incluido de manera que podemos tener acceso al tipo de señal de salida. Las funciones serán explicadas mas tarde cuando sus callbacks sean conectados.

Example 4.7. Cliente Ecore_Ipc: inicio en main

```
int main(int argc, char ** argv) {
    Ecore_Ipc_Server *server;

    if (!ecore_init()) {
        printf("unable to init ecore\n");
        return 1;
    }

    if (!ecore_ipc_init()) {
        printf("unable to init ecore_con\n");
        ecore_shutdown();
        return 1;
    }
    ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, sig_exit_cb, NULL);
```

Como mencionado anteriormente, pese a que estamos escribiendo una aplicación cliente, aún usamos un objeto Ecore_Ipc_Server. Usar Ecore_Ipc requiere la inicialización de Ecore. Esto se hace con una simple llamada a `ecore_init`. Ecore_Ipc es entonces iniciado con una llamada a `ecore_ipc_init`. Si alguna de estas devuelve 0, la acción apropiada es tomada para deshacer cualquier inicialización hecha hasta este punto. El callback ECORE_EVENT_SIGNAL_EXIT es conectado de manera que podemos salir de manera apropiada si es requerido.

Example 4.8. Cliente Ecore_Ipc: main creando cliente

```
    server = ecore_ipc_server_connect(ECORE_IPC_REMOTE_SYSTEM,
                                      "localhost", 9999, NULL);
    ecore_event_handler_add(ECORE_IPC_EVENT_SERVER_ADD,
                                      handler_server_add, NULL);
    ecore_event_handler_add(ECORE_IPC_EVENT_SERVER_DEL,
                                      handler_server_del, NULL);
    ecore_event_handler_add(ECORE_IPC_EVENT_SERVER_DATA,
                                      handler_server_data, NULL);
```

En este ejemplo creamos una conexión remota al servidor llamado "localhost" en el puerto 9999. Esto es hecho con el método `ecore_ipc_server_connect`. El primer parámetro es el tipo de conexión que se hace, que puede ser una de: ECORE_IPC_REMOTE_SYSTEM, ECORE_IPC_LOCAL_SYSTEM, o ECORE_IPC_LOCAL_USER. Si OpenSSL estaba disponible cuando Ecore_Ipc fue compilado, ECORE_IPC_USE_SSL puede ser combinado con el tipo de conexión por medio de un OR, para crear una conexión SSL.

Las tres llamadas a `ecore_event_handler_add` configuran los callbacks para los distintos tipos de eventos que iremos recibiendo del servidor. Un servidor fue añadido, un servidor fue eliminado, o el servidor nos envió datos.

Example 4.9. Cliente Ecore_Ipc: final de main

```
    ecore_ipc_server_send(server, 3, 4, 0, 0, 0, "Look ma, no pants", 17);

    ecore_main_loop_begin();

    ecore_ipc_server_del(server);
    ecore_ipc_shutdown();
    ecore_shutdown();
    return 0;
}
```

Para el propósito de este ejemplo, el cliente está enviando un mensaje al servidor al iniciarse, al cual el servidor responderá. El mensaje del cliente es enviado con `ecore_ipc_server_send`. `ecore_ipc_server_send` toma el servidor al que enviar, el número mayor y menor del mensaje, una referencia, dos parámetros ref y ref_to, una respuesta, los datos y el tamaño. Estos parámetros, excepto por el servidor, son decididos por el cliente y pueden referirse a cualquier cosa que se requiera. Esto intenta dar máxima flexibilidad creando aplicaciones IPC cliente/servidor.

Tras enviar el mensaje entramos en el ciclo principal ecore y esperamos eventos. Si salimos del ciclo principal eliminamos el objeto servidor, finalizamos Ecore_Ipc con una llamada a `ecore_ipc_shutdown`, y finalizamos ecore con `ecore_shutdown`.

Example 4.10. Cliente Ecore_Ipc: sig_exit_cb

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

`sig_exit_cb` simplemente dice a ecore que salga del ciclo principal. Esto no es estrictamente necesario porque si la única cosa siendo hecha es llamar a `ecore_main_loop_quit()`, Ecore manejará esto por si mismo si no hay manejador. Pero esto muestra como un manejador puede ser creado si se necesita en la aplicación.

Example 4.11. Cliente Ecore_Ipc: los callbacks

```
int handler_server_add(void *data, int ev_type, void *ev) {
    Ecore_Ipc_Event_Server_Add *e = (Ecore_Ipc_Event_Server_Add *)ev;
    printf("Got a server add %p\n", e->server);
    return 1;
}

int handler_server_del(void *data, int ev_type, void *ev) {
    Ecore_Ipc_Event_Server_Del *e = (Ecore_Ipc_Event_Server_Del *)ev;
    printf("Got a server del %p\n", e->server);
    return 1;
}

int handler_server_data(void *data, int ev_type, void *ev) {
    Ecore_Ipc_Event_Server_Data *e = (Ecore_Ipc_Event_Server_Data *)ev;
    printf("Got server data %p [%i] [%i] [%i] (%s)\n", e->server, e->major,
                                e->minor, e->size, (char *)e->data);
    return 1;
}
```

Estos tres callbacks, `handler_server_add`, `handler_server_del`, y `handler_server_data`, son cuerpo del cliente manejando todos los eventos relacionados con el servidor al que estamos conectados. Cada uno de los callbacks tiene asociada una estructura de eventos Ecore_Ipc_Event_Server_Add, Ecore_Ipc_Event_Server_Del y Ecore_Ipc_Event_Server_Data, conteniendo información sobre el evento.

Cuando nos conectemos por primera vez al servidor la función callback `handler_server_add` será ejecutada permitiendo llevar a cabo cualquier inicialización.

Si el servidor corta la conexión, `handler_server_del` será ejecutada permitiendo cualquier limpieza adicional.

Cuando el servidor envía datos al cliente, `handler_server_data` será ejecutada. Lo cual en este ejemplo imprime alguna información sobre el propio mensaje y su cuerpo.

Y eso es el cliente. El código es bastante simple gracias a las abstracciones que provee Ecore.

Example 4.12. Servidor Ecore_Ipc : preámbulo

```
#include <Ecore.h>
#include <Ecore_Ipc.h>

int sig_exit_cb(void *data, int ev_type, void *ev);
int handler_client_add(void *data, int ev_type, void *ev);
int handler_client_del(void *data, int ev_type, void *ev);
int handler_client_data(void *data, int ev_type, void *ev);
```

Como con el cliente, la cabecera Ecore.h es incluida para conseguir acceso a la señal de exit. La cabecera Ecore_Ipc.h es requerida para aplicaciones que usan la librería Ecore_Ipc. Cada manejador de señal será explicado con su código.

Example 4.13. Servidor Ecore_Ipc: inicio de main

```
int main(int argc, char ** argv) {
    Ecore_Ipc_Server *server;

    if (!ecore_init()) {
        printf("Failed to init ecore\n");
        return 1;
    }

    if (!ecore_ipc_init()) {
        printf("failed to init ecore_con\n");
        ecore_shutdown();
        return 1;
    }

    ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, sig_exit_cb, NULL);
```

Esto es lo mismo que la inicialización del cliente mas arriba.

Example 4.14. Servidor Ecore_Ipc: main creando el servidor

```
    server = ecore_ipc_server_add(ECORE_IPC_REMOTE_SYSTEM, "localhost", 9999, NULL
    ecore_event_handler_add(ECORE_IPC_EVENT_CLIENT_ADD, handler_client_add, NULL);
    ecore_event_handler_add(ECORE_IPC_EVENT_CLIENT_DEL, handler_client_del, NULL);
    ecore_event_handler_add(ECORE_IPC_EVENT_CLIENT_DATA, handler_client_data, NULL
```

A diferencia del cliente, para el servidor añadimos un oyente al puerto 9999 en la máquina "localhost" por medio de la llamada a `ecore_ipc_server_add`. Esto creará y nos devolverá el objeto servidor. Entonces conectamos los manejadores de eventos requeridos, la diferencia con el cliente siendo que esta vez queremos eventos CLIENT en vez de eventos SERVER.

Example 4.15. Cliente Ecore_Ipc client: final de main

```
    ecore_main_loop_begin();

    ecore_ipc_server_del(server);
    ecore_ipc_shutdown();
    ecore_shutdown();
    return 0;
}
```

De nuevo esto es idéntico a la finalización del cliente, menos el envío de datos al servidor.

Example 4.16. Servidor Ecore_Ipc: callback sig_exit

`sig_exit_cb` es de nuevo identica a la vista en el cliente.

Example 4.17. Servidor Ecore_Ipc: los callbacks

```
int handler_client_add(void *data, int ev_type, void *ev) {
    Ecore_Ipc_Event_Client_Add *e = (Ecore_Ipc_Event_Client_Add *)ev;
    printf("client %p connected to server\n", e->client);
    return 1;
}

int handler_client_del(void *data, int ev_type, void *ev) {
    Ecore_Ipc_Event_Client_Del *e = (Ecore_Ipc_Event_Client_Del *)ev;
    printf("client %p disconnected from server\n", e->client);
    return 1;
}

int handler_client_data(void *data, int ev_type, void *ev) {
    Ecore_Ipc_Event_Client_Data *e = (Ecore_Ipc_Event_Client_Data *)ev;
    printf("client %p sent [%i] [%i] [%i] (%s)\n", e->client, e->major,
                                  e->minor, e->size, (char *)e->data);

    ecore_ipc_client_send(e->client, 3, 4, 0, 0, 0, "Pants On!", 9);
    return 1;
}
```

Los callbacks de evento son similares a los vistos en la aplicación cliente. La principal diferencia es que los eventos son eventos _Client_ en vez de eventos _Server_.

El callback add es cuando un cliente se conecta a nuestro servidor, siendo el callback del su opuesto cuando el cliente desconecta. El callback data es para cuando un cliente envía datos al servidor.

Al final del callback `handler_client_data` hacemos una llamada a `ecore_ipc_client_send`. Esto envía datos al cliente. Como con el envío de datos al servidor, los parámetros son: el cliente al que enviar, numero mayor y menor, ref , ref_to, respuesta, datos y tamaño de datos.

Example 4.18. Ecore_Ipc: compilation

```
CC = gcc

all: server client

server: server.c
    $(CC) -o server server.c `ecore-config --cflags --libs`

client: client.c
    $(CC) -o client client.c `ecore-config --cflags --libs`

clean:
    rm server client
```

Como con otras aplicaciones ecore, es muy fácil compilar una aplicación Ecore_Ipc. Si tu Ecore fue

compilado con Ecore_Ipc, simplemente invocar el comando "ecore-config --cflags --libs añadirá todos los caminos de librería e información de enlace que sea requerida.

Como se ha visto en este ejemplo, Ecore_Ipc es una librería facil de usar para crear aplicaciones cliente/servidor.

Receta:Temporizadores Ecore

dan 'dj2' sinclair <zero@perplexity.org>

Si necesitas que un callback se accione en un determinado momento, con posibilidad de repetir el temporizador continuamente, entonces `Ecore_Timer` es lo que estás buscando.

Example 4.19. Temporizadores Ecore

```
#include <stdio.h>
#include <Ecore.h>

static int timer_one_cb(void *data);
static int timer_two_cb(void *data);

int main(int argc, char ** argv) {
    ecore_init();

    ecore_timer_add(1, timer_one_cb, NULL);
    ecore_timer_add(0.5, timer_two_cb, NULL);

    ecore_main_loop_begin();
    ecore_shutdown();

    return 0;
}

static int timer_one_cb(void *data) {
    printf("1");
    fflush(stdout);
    return 1;
}

static int timer_two_cb(void *data) {
    printf("2");
    fflush(stdout);
    return 1;
}
```

La creación de los temporizadores es tan simple como llamar a `ecore_timer_add()`. Esto devolverá un `Ecore_Timer` o NULL en caso de fallo. En este caso estoy ignorando el valor de retorno. Los tres parámetros son:

- double timeout
- int (*callback)(void *data)
- const void *user_data

El timeout da el numero de segundos en que expirará este temporizador. En el caso de este ejemplo le damos 1 segundo y 0.5 segundos respectivamente. La función callback es la que será ejecutada cuando

el temporizador expire y user_data son los datos que pasar a la función callback.

Las funciones callback tienen todas la misma firma `int callback(void *data)`. El valor de retorno del temporizador debiera ser 0 o 1. Si devuelves 0 el temporizador expirará y *no* se ejecutará otra vez. Si devuelves 1, el temporizador será programado para re-ejecutarse en el periodo de tiempo dado por el timeout. Esto te permite activar o continuar el temporizador como requiera tu programa.

Si tienes un temporizador que deseas eliminar en algun punto del futuro puedes llamar a `ecore_timer_del(Ecore_Timer *)`. Si la eliminación se lleva a cabo con éxito el puntero será devuelto, de otra manera NULL será devuelto. Despues de llamar a la función de borrado la estructura `Ecore_Timer` será invalida y no deberías usarla de nuevo en tu programa.

Compilar el ejemplo es tan simple como:

Example 4.20. Compilación

```
gcc -Wall -o etimer etimer.c `ecore-config --cflags --libs`
```

Si ejecutas el programa deberías ver una serie de unos y doses en pantalla con el doble de doses que de unos.

Los temporizadores `Ecore_Timer` son fáciles de iniciar y usar y proveen un poptente mecanismo de temporización a tus programas.

Chapter 5. EDB & EET

EDB es una librerÍa de conveniencia de base de datos envolviendo Berkeley DB 2.7.7 por Sleepycat Software. Su intenciÓn es hacer fácil, rápido'eficiente y portable el acceder a informaciÓn de bases de datos.

EET es una librerÍa mínima diseñada para escribir trozos arbitrarios de datos a un archivo y opcionalmente comprime cada trozo (de manera semejante a un archivo zip) y permite una veloz lectura de acceso aleatorio mas tarde. No hace un archivo zip, dado que un zip tiene mas complejidad de la necesaria, y fue mas simple implementar esto una vez aquí.

EDB provee un método excelente de almacenar y recuperar información de configuración de la aplicación, aunque puede ser usado de una manera mucho mas extensiva que esa. Ebits, el predecesor de Edje, incluso usaba EDB como contenedor para temas Ebits previamente a EET. Una Edb consiste de una serie de pares valor/clave, que pueden consistir de una variedad de tipos de datos, incluyendo enteros, valores en punto flotante, cadenas, y datos binarios. La API simplificada provee funciones simples, completas, y unificadas para gestionar y acceder tu base de datos.

Adicionalmente a la librerÍa, hay disponible una variedad de herramientas para acceder y modificar tus EDBs. edb_ed provee un simple interfaz de linea de comandos que puede ser fácilmente programado en scripts, especialmente útil para uso con GNU autotools. edb_vt_ed provee un interfaz ncurses fácil de usar. Finalmente, edb_gtk_ed provee un interfaz elegante y sencillo, especialmente útil para la edición por el usuario final de los datos contenidos en EDBs.

Eet es extremadamente rápida, pequeña y simple. Los archivos Eet pueden ser pequeños y altamente comprimidos, haciendolos óptimos para enviar por internet sin tener que archivar, comprimir o descomprimir, e instalarlos. Permiten lecturas de acceso aleatorio veloces como el rayo una vez creados, haciendolos perfectos para almacenar datos que son escritos una sola vez (o raramente) y leidos muchas veces, pero el programa no quiere tener que leerlo todo a la vez.

Tambien puede codificar y descodificar estructuras de datos en memoria, así como datos de imagen para grabar a archivos Eet o para enviar por la red a otras máquinas, o simplemente escribir a archivos arbitrarios en el sistema. Todos los datos son codificados en manera independiente de la plataforma y pueden ser leídos y escritos por cualquier arquitectura.

Receta: Creando archivos EDB desde la línea de comandos

dan 'dj2' sinclair <zero@perplexity.org>

A menudo es deseable crear los archivos EDB desde una simple script de línea de comandos. entonces puede ser hecha parte del proceso de construcción.

Esto puede llevarse a cabo fácilmente con el programa **edb_ed**. **edb_ed** es un interfaz muy simple para EDB, permitiéndote crear/editar/borrar pares valor/clave dentro de bases de datos EDB.

Example 5.1. script de línea de comandos EDB

```
#!/bin/sh

DB=out.db

edb_ed $DB add /global/debug_lvl int 2
```

```
edb_ed $DB add /foo/theme str "default"
edb_ed $DB add /bar/number_of_accounts int 1
edb_ed $DB add /nan/another float 2.3
```

Si el archivo de output no existe la primera vez que se llame a un comando `add`, entonces **edb_ed** creará el archivo y hará cualquier inicialización necesaria. `add` es usado para añadir entradas en la base de datos. El primer parámetro despues de add es la clave con la cual los datos serán insertados en la base de datos. Esta clave será usada por tu aplicación en el futuro para actualizar los datos. El siguiente parámetro es el tipo de los datos que se van a añadir. Esto puede ser uno de:

- int
- str
- float
- data

El ultimo parámetro es el valor que ha de ser asociado con esta clave.

Usando **edb_ed** puedes crear/editar/ver cualquier archivo EDB requerido por tu aplicación de manera fácil y rápida.

Receta: introducción a EDB

dan 'dj2' sinclair `<zero@perplexity.org>`

EDB provee un potente backend de base de datos para usar en tu aplicación. Esta receta es una simple introducción que abrirá una base de datose, escribirá varias claves y las volverá a leer.

Example 5.2. introducción a EDB

```
#include <stdio.h>
#include <Edb.h>

#define INT_KEY     "/int/val"
#define STR_KEY     "/str/val"
#define FLT_KEY     "/float/val"

int main(int argc, char ** argv) {
    E_DB_File *db_file = NULL;
    char *str;
    int i;
    float f;

    if (argc < 2) {
        printf("Need db file\n");
        return 0;
    }

    db_file = e_db_open(argv[1]);
    if (db_file == NULL) {
        printf("Error opening db file (%s)\n", argv[1]);
        return 0;
    }

    printf("Adding values...\n");
    e_db_int_set(db_file, INT_KEY, 42);
```

```
    e_db_float_set(db_file, FLT_KEY, 3.14159);
    e_db_str_set(db_file, STR_KEY, "My cats breath smells like...");

    printf("Reading values...\n");
    if (e_db_int_get(db_file, INT_KEY, &i))
        printf("Retrieved (%s) with value (%d)\n", INT_KEY, i);

    if (e_db_float_get(db_file, FLT_KEY, &f))
        printf("Retrieved (%s) with value (%f)\n", FLT_KEY, f);

    if ((str = e_db_str_get(db_file, STR_KEY)) != NULL) {
        printf("Retrieved (%s) with value (%s)\n", STR_KEY, str);
        free(str);
    }

    e_db_close(db_file);
    e_db_flush();

    return 1;
}
```

Para usar WDB debes incluir <Edb.h> en tu archivo para tener acceso a la API. El ptincipio del programa es bastante estándar. Tengo una tendencia a hacer errores tecleando así que definí las diferentes claves que iré usando. En cuanto tenemos el nombre de archivo intentamos abrir/crear la base de datos.

La base de datos será abierta, o si no existe, creada con la llamada a `e_db_open()` que devolverá NULL en caso de encontrar algún error.

Una vez la base de datos ha sido abierta podemos excribir nuestros valores. Esto se hace por medio de las tres llamadas: `e_db_int_set()`, `e_db_float_set()` y `e_db_str_set()`.Tambien puedes introducir datos genéricos en un archivo de base de datos con `e_db_data_set()`.

Junto con datos normales, puedes introducir metadatos sobre la base de datos en el propio archivo. Estos datos no se pueden ser recuperados con los métodos normales get/set. Estas propiedades se asignan con `e_db_property_set()`

Cada uno de los métodos de asignación de tipos toma tres parámetros:

- E_DB_File *db
- char *key
- value

El parámetro `value` es del tipo correspondiente al método, int, float, char * or void * for _int_set, _float_set, _str_set y _data_set respectivamente.

Una vez los valores están en la base de datos pueden ser recuperados con los métodos de acceso. Cada uno de estos métodos toma tres parámetros y devuelve un int. El valor de retorno es 1 en caso de acceso con éxito y 0 en otro caso.

Como con los métodos de asignación, los parámetros de los métodos de acceso son la base de datos, la clave, y un puntero al lugar donde escribir el valor.

En cuanto terminemos con la base de datos podemos cerrarla con una llamada a `e_db_close()`. La llamada a `e_db_close()` no garantiza que la base de datos haya sido escrita a disco, para esto llamamos a `e_db_flush()` que escribirá los contenidos a disco.

Example 5.3. Compiling

```
zero@oberon [edb] -> gcc -o edb edb_ex.c \
        `edb-config --cflags --libs`
```

Si ejecutas el programa deberías ver los valores imprimidos a la pantalla, y despues de la ejecución habrá un archivo .db con el nombre especificado. Entonces puedes echar un vistazo a la base de datos con **edb_gtk_ed** y ver los valores introducidos.

Receta: recuperación de claves EDB

dan 'dj2' sinclair <zero@perplexity.org>

La API EDB hace una tarea simple el recuperar todas las claves disponibles en la base de datos. Estas claves pueden entonces ser usadas para determinar los tipos del objeto en la base de datos, o simplemente para recuperar el objeto.

Example 5.4. recuperación de claves EDB

```
#include <Edb.h>

int main(int argc, char ** argv) {
    char ** keys;
    int num_keys, i;

    if (argc < 2)
        return 0;

    keys = e_db_dump_key_list(argv[1], &num_keys);
    for(i = 0; i < num_keys; i++) {
        printf("key: %s\n", keys[i]);
        free(keys[i]);
    }
    free(keys);
    return 1;
}
```

Recuperar las claves se hace simplemente mediante la llamada a `e_db_dump_key_list()`. Esto devolverá un array de cadenas clave char **. Estas cadenas, y el propio array, debe ser liberado (free) por la aplicación. `e_db_dump_key_list()` devolverá tambien el número de claves en el array en el parámetro `num_keys`.

Notarás que no necesitamos abrir la base de datos para llamar a `e_db_dump_key_list()`. Esta función trabaja en el archivo en sí mismo, en lugar de en un objeto db.

Example 5.5. Compilación

```
zero@oberon [edb] -> gcc -o edb edb_ex.c \
```

```
`edb-config --cflags --libs`
```

Ejecutar el programa debería producir un listado de todas las claves en la base de datos dada. Esto puede ser verificado mirando la base de datos con una herramienta externa como **edb_gtk_ed**.

Chapter 6. Esmart

Esmart provee una variedad de objetos inteligentes EVAS que proveen una potencia significativa a tus aplicaciones basadas en EVAS y EFL.

Receta: Introducción a Esmart_Trans

dan 'dj2' sinclair <zero@perplexity.org>

Transparencia está convirtiendose cada vez más en un rasgo común de las aplicaciones. Con este objeto, el objecto Esmart_Trans ha sido creado. Este objeto hará todo el trabajo duro para producir un fondo transparente para tu programa.

Esmart_Trans hace muy facil la integración de un fondo transparente en tu aplicación. Necesitas crear el objeto trans, y entonces asegurarte de actualizarlo conforme la ventana sea movida o redimensionada.

Example 6.1. Includes y declaraciones

```
#include <stdio.h>
#include <Ecore.h>
#include <Ecore_Evas.h>
#include <Esmart/Esmart_Trans_X11.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();
```

Cada aplicación que use un objeto Esmart_Trans va a necesitar Ecore, Ecore_Evas, y los archivos de cabecera de Esmart/Esmart_Trans. Las cuatro declaraciones siguientes son callbacks desde ecore para eventos en nuestra ventana, salida, borrado, redimensionado, y movimiento respectivamente. Las últimas dos declaraciones son funciones de conveniencia que se usan en el ejemplo y no necesitan estar en tu programa.

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;
}
```

La rutina principal para este programa es bastante simple. Ecore y Ecore_Evas son ambos inicializados, con la apropiada comprobación de errores. Entonces creamos el gui y empezamos el ciclo de eventos principal de ecore. Cuando ecore sale cerramos todo y retornamos.

Example 6.3. callbacks de salida y borrado

```
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();
}
```

Los callbacks de salida y borrado son los callbacks ecore genéricos. El callback de salida no es estrictamente necesario, dado que Ecore llamará a `ecore_main_loop_quit()` si no hay ningun manejador registrado, pero se incluye para mostrar como se hace.

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);
}
```

La rutina `_freshen_trans` es una rutina de ayude para actualizar la imagen a la que la transparencia

se muestra. Esta será llamada cuando necesitemos actualizar nuestra imagen a lo que está en ese momento bajo la ventana. La función captura el tamaño actual del ecore_evas, y entonces adquiere el objeto con el nombre "bg" (este nombre es el mismo que damos a nuestra trans cuando la creamos).Entonces, siempre que el objeto exista, le decimos a esmart que refresque la imagen siendo mostrada.

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);
}
```

Cuando la ventana sea redimensionada necesitamos redimensionar nuestro evas al tamaño correcto y entonces actualizar el objeto trans para mostrar ese pedazo de background. Capturamos el tamaño actual de la ventana con `ecore_evas_geometry_get` y el tamaño máximo y mínimo de la ventana. Mientras nuestro tamaño deseado esté entre los límites máximo y mínimo de la ventana, capturamos el objeto "bg" (de nuevo el mismo que el título) y lo redimensionamos. Una vez hemos hecho el redimensionado, llamamos a la rutina `_freshen_trans` para actualizar la imagen mostrada en el fondo.

Example 6.6. move_cb

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

Cuando la ventana es movida necesitamos refrescar la imágen mostrada como transparencia.

Example 6.7. Iniciar 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);
```

La primera porción de make_gui se encarga de iniciar ecore y ecore_evas. Primero al callback de salida es conectado en ECORE_EVENT_SIGNAL_EXIT, entonces el objeto Ecore_Evas es creado con el motor software X11. Se inicia el título de la ventana y conectamos los callbacks escritos arriba, delete, resize y move. Finalmente capturamos el evas para el Ecore_Evas creado.

Example 6.8. Creando el objeto Esmart_Trans

```
    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);
}
```

Una vez todo está iniciado podemos crear el objeto trans. El trans ha de ser creado en el evas devuelto por `ecore_evas_get`. Esta creación inicial es hecha mediante la llamada a `esmart_trans_x11_new`. En cuanto tenemos el objeto, lo movemos de manera que empiece en la posición (0 , 0) (la esquina superior izquierda), iniciamos la capa a -5 y llamamos al objeto "bg" (como usamos mas arriba). Entonces capturamos el tamaño actual del ecore_evas y lo usamos para redimensionar el objeto trans al tamaño de la ventana. Cuando todo ha sido redimensionado mostramos trans y el ecore_evas. Como paso final, refrescamos la imagen en la transparencia a la que está actualmente bajo la ventana de manera que esté actualizada.

Example 6.9. makefile sencilla

```
CFLAGS = `ecore-config --cflags` `evas-config --cflags` `esmart-config --cflags`
LIBS = `ecore-config --libs` `evas-config --libs` `esmart-config --libs` \
            -lesmart_trans_x11

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

Para compilar el programa de arriba necesitamos incluir la información de librería para ecore,

ecore_evas y esmart. Esto es hecho por medio de las scripts -config para cada librería. Estas scripts -config saben donde está cada uno de los includes y librerías e inicia los paths de include y enlazado apropiados para la compilación.

Receta: Introducción al contenedor Esmart

dan 'dj2' sinclair <zero@perplexity.org>

Usualmente mientras se diseña el UI de una aplicación hay un deseo de agrupar elementos comunes juntos y hacer que su colocación dependa el uno del otro. Con este objeto la librería Esmart Container ha sido creada. Ha sido diseñada para manejar las partes dificiles de la colocación, y en casos donde no hace lo que necesitas, las porciones de colocación del contenedor son extensibles y cambiables.

Esta receta dará la base de usar un container Esmart. El producto final es un programa que te permitirá ver algunas de las diferentes combinaciones de disposición del container por defecto. La colocación será hecha por Edje con callbacks al programa para cambiar la disposición del contenedor, y para decir si el usuario pulsó en un elemento del container.

Example 6.10. Includes y declaraciones

```
#include <Ecore.h>
#include <Ecore_Evas.h>
#include <Edje.h>
#include <Esmart/Esmart_Container.h>
#include <getopt.h>

static void make_gui(const char *theme);
static void container_build(int align, int direction, int fill);
static void _set_text(int align, int direction);
static void _setup_edje_callbacks(Evas_Object *o);
static void _right_click_cb(void* data, Evas_Object* o, const char* emmission,
                                                        const char* source);
static void _left_click_cb(void* data, Evas_Object* o, const char* emmission,
                                                        const char* source);
static void _item_selected(void* data, Evas_Object* o, const char* emmission,
                                                        const char* source);

static Ecore_Evas *ee;
static Evas_Object *edje;
static Evas_Object *container;

char *str_list[] = {"item 1", "item 2",
                    "item 3", "item 4",
                    "item 5"};
```

Como con otros programas EFL necesitamos incluir Ecore, Ecore?Evas, Edje y dado que este es un ejemplo de contenedor, la cabecera de Esmart/Esmart_Container. Getopt será usado para permitir algún procesado de la línea de comandos.

Despues vienen los prototipos de funciones que serán descritas mas tarde cuando lleguemos a sus implementaciones. Entonces unas cuantas variables globales que serñan usadas durante el programa. El array str_list es el contenido a ser almacenado en el contenedor.

Example 6.11. main

```
int main(int argc, char ** argv) {
    int align = 0;
    int direction = 0;
    int fill = 0;
    int ret = 0;
    int c;
    char *theme = NULL;

    while((c = getopt(argc, argv, "a:d:f:t:")) != -1) {
        switch(c) {
            case 'a':
                align = atoi(optarg);
                break;

            case 'd':
                direction = atoi(optarg);
                break;

            case 'f':
                fill = atoi(optarg);
                break;

            case 't':
                theme = strdup(optarg);
                break;

            default:
                printf("Unknown option string\n");
                break;
        }
    }

    if (theme == NULL) {
        printf("Need a theme defined\n");
        exit(-1);
    }
```

El principio de la funciópn principal extrae las opciones de los argumentos de la linea de comandos e inicia la pantalla por defecto. Como puedes ver, requerimos un tema para mostrar. Esto podría ser hecho mas inteligente, buscando directorios de instalación por defecto y los directorios de aplicación del usuario, pero este ejemplo opta por la via facil y fuerza el tema a ser una opción de la linea de comandos.

Example 6.12. Inicialización

```
    if (!ecore_init()) {
        printf("Can't init ecore, bad\n");
        ret = 1;
        goto EXIT;
    }
    ecore_app_args_set(argc, (const char **)argv);

    if (!ecore_evas_init()) {
        printf("Can't init ecore_evas, bad\n");
        ret = 1;
        goto EXIT_ECORE;
    }
```

```
    if (!edje_init()) {
        printf("Can't init edje, bad\n");
        ret = 1;
        goto EXIT_ECORE_EVAS;
    }
    edje_frametime_set(1.0 / 60.0);

    make_gui(theme);
    container_build(align, direction, fill);

    ecore_main_loop_begin();
```

Tras recibir los argumentos de la linea de comandos, procedemos a inicializar las librerías requeridas, Ecore, Ecore_Evas y Edje. Tomamos el paso adicional de iniciar el frame rate de Edje.

En cuanto la inicialización está completada creamos el GUI inicial para la aplicación. He separado la construcción del contenedor en una función separada para hacer el código del contenedor mas facil de localizar en el ejemplo.

Cuando todo está creado llamamos a `ecore_main_loop_begin` y esperamos a que ocurran eventos.

Example 6.13. Finalización

```
    edje_shutdown();

EXIT_ECORE_EVAS:
    ecore_evas_shutdown();

EXIT_ECORE:
    ecore_shutdown();

EXIT:
    return ret;
}
```

La rutina de finalización habitual, seamos buenos programadores y cerremos todo lo que empezamos.

Example 6.14. callbacks de ventana

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

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

static 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), "edje")))
                evas_object_resize(o, w, h);
        }
    }
}
```

Lo siguiente iniciamos algunos callbacks genéricos para ser usados por el UI. Estos serán los callbacks de exit, destroy y resize. De nuevo, las funciones habituales EFL. Aunque el callback de salida no es estrictamente necesario dado que Ecore llamará a `ecore_main_loop_quit()` si no hay ningun manejador registrado para este callback.

Example 6.15. make_gui

```
static void make_gui(const char *theme) {
    Evas *evas = NULL;
    Evas_Object *o = NULL;
    Evas_Coord minw, minh;

    ee = NULL;
    edje = NULL;
    container = NULL;

    ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, sig_exit_cb, NULL);

    ee = ecore_evas_software_x11_new(NULL, 0, 0, 0, 300, 400);
    ecore_evas_title_set(ee, "Container Example");

    ecore_evas_callback_delete_request_set(ee, win_del_cb);
    ecore_evas_callback_resize_set(ee, win_resize_cb);
    evas = ecore_evas_get(ee);

    // create the edje
    edje = edje_object_add(evas);
    evas_object_move(edje, 0, 0);

    if (edje_object_file_set(edje, theme, "container_ex")) {
        evas_object_name_set(edje, "edje");

        edje_object_size_min_get(edje, &minw, &minh);
        ecore_evas_size_min_set(ee, (int)minw, (int)minh);
        evas_object_resize(edje, (int)minw, (int)minh);
        ecore_evas_resize(ee, (int)minw, (int)minh);

        edje_object_size_max_get(edje, &minw, &minh);
        ecore_evas_size_max_set(ee, (int)minw, (int)minh);
        evas_object_show(edje);

    } else {
        printf("Unable to open (%s) for edje theme\n", theme);
```

```
        exit(-1);
    }
    _setup_edje_callbacks(edje);
    ecore_evas_show(ee);
}
```

El GUI consiste del Ecore_Evas que contiene el propio canvas, y el Edje que usaremos para controlar nuestra disposición. La función `make_gui` inicia los callbacks definidos mas arriba y crea el Ecore_Evas.

En cuanto tenemos el Evas y los callbacks están definidos, creamos el objeto Edje que definirá nuestra colocación. La llamada a `edje_object_add` es usada para crear el objeto en el Evas, y cuando eso ha sido hecho, tomamos el tema pasado por el usuario y configuramos Edje para usar dicho tema, el parámetro "container_ex" es el nombre del grupo dentro del EET que hemos de usar.

Una vez el archivo de tema es en el Edje, usamos los valores en el archivo de tema para iniciar los rangos de tamaño de la aplicación, y mostramos el Edje. Entonces iniciamos los callbacks en el Edje y enseñamos el Ecore_Evas.

Example 6.16. Callbacks Edje

```
static void _setup_edje_callbacks(Evas_Object *o) {
    edje_object_signal_callback_add(o, "left_click",
                        "left_click", _left_click_cb, NULL);
    edje_object_signal_callback_add(o, "right_click",
                        "right_click", _right_click_cb, NULL);
}
```

El programa tendrá dos callbacks principales conectados al Edje, uno para la señal de pulsación de botón izquierdo y otro para la señal del derecho. Estos serán usados para cambiar la dirección/alineación del contenedor. Los parámetros segundo y tercero de los callbacks necesitan encajar con los datos emitidos con la señal desde Edje, esto será visto mas tarde cuando miremos el archivo EDC. El tercer parámetro es la función a llamar, y el último, cualesquiera datos que queramos pasar al callback.

Example 6.17. container_build

```
static void container_build(int align, int direction, int fill_policy) {
    int len = 0;
    int i = 0;
    const char *edjefile = NULL;

    container = esmart_container_new(ecore_evas_get(ee));
    evas_object_name_set(container, "the_container");
    esmart_container_direction_set(container, direction);
    esmart_container_alignment_set(container, align);
    esmart_container_padding_set(container, 1, 1, 1, 1);
    esmart_container_spacing_set(container, 1);
    esmart_container_fill_policy_set(container, fill_policy);

    evas_object_layer_set(container, 0);
    edje_object_part_swallow(edje, "container", container);
```

La función `container_build` creará el contenedor y iniciará nuestros elementos de datos en el container. La creación es lo suficientemente facil con una llamada a `esmart_container_new` devolvioendo el Evas_Object que es el contenedor. Una vez el contenedor es creado podemos dar un nombre al container para hacer la referencia mas facil.

Lo siguiente, iniciamos la dirección, que es CONTAINER_DIRECTION_VERTICAL o CONTAINER_DIRECTION_HORIZONTAL [o en este caso, un int siendo pasado desde la línea de comandos dado que las dos direcciones se traducen a 1 y 0 respectivamente]. La dirección dice al contenedor de que manera serán dibujados los elementos.

Despues de la dirección iniciamos la alineación del contenedor. La alineación dice al container donde dibujar los elementos. Los valores posibles son: CONTAINER_ALIGN_CENTER, CONTAINER_ALIGN_LEFT, CONTAINER_ALIGN_RIGHT, CONTAINER_ALIGN_TOP and CONTAINER_ALIGN_BOTTOM. Con la colocación por defecto, derecha e izquierda se aplican solo a un contenedor vertical, y arriba y abajo se aplican solo a un contenedor horizontal, aunque centro se aplica a ambos.

Si quisieramos usar un esquema de disposición diferente al que hay por defecto, podríamos poner una llamada a `esmart_container_layout_plugin_set(container, "name")` donde el nombre es el nombre del plugin a usar. La configuración por defecto es el contenedor llamado "default".

En cuanto las direcciones y alineación están iniciadas, el espaciado y rellenado del contenedor son especificados. El rellenado especifica el espacio alrededor del contenedor tomando cuatro parámetros: left, right, top y bottom. El parámetro de espaciado especifica el espacio entre elementos en el contenedor.

Entonces continuamos e iniciamos la política de relleno del contenedor. Esto especifica como los elementos son posicionados para rellenar el espacio en el contenedor. Los posibles valores son: CONTAINER_FILL_POLICY_NONE, CONTAINER_FILL_POLICY_KEEP_ASPECT, CONTAINER_FILL_POLICY_FILL_X, CONTAINER_FILL_POLICY_FILL_Y, CONTAINER_FILL_POLICY_FILL and CONTAINER_FILL_POLICY_HOMOGENOUS.

Una vez que el contenedor está completamente especificado iniciamos la capa del contenedor, y tragamos el contenedor en el edje y la parte llamada "container".

Example 6.18. Añadiendo Elementos al Contenedor

```
len = (sizeof(str_list) / sizeof(str_list[0]));
for(i = 0; i < len; i++) {
    Evas_Coord w, h;
    Evas_Object *t = edje_object_add(ecore_evas_get(ee));

    edje_object_file_get(edje, &edjefile, NULL);
    if (edje_object_file_set(t, edjefile, "element")) {
        edje_object_size_min_get(t, &w, &h);
        evas_object_resize(t, (int)w, (int)h);

        if (edje_object_part_exists(t, "element.value")) {
            edje_object_part_text_set(t, "element.value", str_list[i]);
            evas_object_show(t);
            int *i_ptr = (int *)malloc(sizeof(int));
            *i_ptr = (i + 1);

            edje_object_signal_callback_add(t, "item_selected",
                                "item_selected", _item_selected, i_ptr);

            esmart_container_element_append(container, t);
        } else {
```

```
                printf("Missing element.value part\n");
                evas_object_del(t);
            }
        } else {
            printf("Missing element part\n");
            evas_object_del(t);
        }
    }
    evas_object_show(container);
    _set_text(align, direction);
}
```

Ahora que tenemos un contenedor, podemos añadir algunos elementos para mostrar. Cada una de las entradas en el array str_list definido al principio del programa será añadida en el contenedor como una parte de texto.

Para cada elemento creamos un nuevo objeto Edje en el Evas. Entonces necesitamos saber el nombre del archivo de tema usado para crear nuestro Edje principal, así que llamamos a `edje_object_file_get` que iniciará el archivo edje a dicho valor.

Entonces intentamos iniciar el grupo llamado "element" en el elemento nuevamente creado. Si esto falla imprimimos un error y borramos el objeto.

En cuanto hayamos encontrado el grupo "element" podemos intentar capturar la parte para nuestro elemento, "element.value". Si esta parte existe, iniciamos el valor de texto de la parte a nuestra cadena actual y mostramos la parte.

Un callback es creado por medio de `edje_object_signal_callback_add` y vinculado al nevo elemento. Este será llamado si la señal "item_selected" es es enviada desde el Edje. El valor i_ptr muestra como se pueden vincular datos al elemento, cuando el usuario hace click en un elemento su número será impreso a la consola.

En cuanto el elemento es creado lo añadimos al contenedor (en este caso, añadiendo el elemento).

Para terminar, el contenedor es mostrado y hacemos algún trabajo extra para mostrar información sobre el contenedor en la cabecera por medio de la llamada `_show_text`.

Example 6.19. _set_text

```
static void _set_text(int align, int direction) {
    Evas_Object *t = edje_object_add(ecore_evas_get(ee));
    const char *edjefile;

    if (direction == CONTAINER_DIRECTION_VERTICAL)
        edje_object_part_text_set(edje, "header_text_direction", "vertical");
    else
        edje_object_part_text_set(edje, "header_text_direction", "horizontal");

    if (align == CONTAINER_ALIGN_CENTER)
        edje_object_part_text_set(edje, "header_text_align", "center");

    else if (align == CONTAINER_ALIGN_TOP)
        edje_object_part_text_set(edje, "header_text_align", "top");

    else if (align == CONTAINER_ALIGN_BOTTOM)
        edje_object_part_text_set(edje, "header_text_align", "bottom");
```

```
    else if (align == CONTAINER_ALIGN_RIGHT)
        edje_object_part_text_set(edje, "header_text_align", "right");

    else if (align == CONTAINER_ALIGN_LEFT)
        edje_object_part_text_set(edje, "header_text_align", "left");
}
```

La rutina `_set_text` toma la dirección y alineación actuales y pone algo de texto en la cabecera del programa. Esta es una comunicación simple con el usuario de la configuración del contenedor actual.

Example 6.20. _left_click_cb

```
static void _left_click_cb(void* data, Evas_Object* o, const char* emmission,
                                                        const char* source) {
    Container_Direction dir = esmart_container_direction_get(container);
    Container_Direction new_dir = (dir + 1) % 2;
    Container_Alignment align = esmart_container_alignment_get(container);

    esmart_container_direction_set(container, new_dir);

    if (align != CONTAINER_ALIGN_CENTER) {
        if (new_dir == CONTAINER_DIRECTION_HORIZONTAL)
            align = CONTAINER_ALIGN_TOP;
        else
            align = CONTAINER_ALIGN_LEFT;
    }
    esmart_container_alignment_set(container, align);
    _set_text(align, new_dir);
}
```

Cuando el usuario pulsa el botón izquierdo del ratón en la pantalla este callback será ejecutado. Tomamos la información actual de dirección del container y conmutamos a la otra dirección (es decir, horizontal se convierte en vertical y viceversa). Tambien reseteamos la alineación si no estamos actualmente alineados al centro para asegurar que todo es válido para la nueva dirección. El texto en la cabecera es actualizado.

Example 6.21. _right_click_cb

```
static void _right_click_cb(void* data, Evas_Object* o, const char* emmission,
                                                        const char* source) {
    Container_Direction dir = esmart_container_direction_get(container);
    Container_Alignment align = esmart_container_alignment_get(container);

    if (dir == CONTAINER_DIRECTION_HORIZONTAL) {
        if (align == CONTAINER_ALIGN_TOP)
            align = CONTAINER_ALIGN_CENTER;

        else if (align == CONTAINER_ALIGN_CENTER)
            align = CONTAINER_ALIGN_BOTTOM;

        else
            align = CONTAINER_ALIGN_TOP;
```

```
    } else {
        if (align == CONTAINER_ALIGN_LEFT)
            align = CONTAINER_ALIGN_CENTER;

        else if (align == CONTAINER_ALIGN_CENTER)
            align = CONTAINER_ALIGN_RIGHT;

        else
            align = CONTAINER_ALIGN_LEFT;
    }
    esmart_container_alignment_set(container, align);
    _set_text(align, esmart_container_direction_get(container));
}
```

El callback de click derecho iterará entre las alineaciones disponibles por una dirección dada cuando el usuario pulse el botón derecho del ratón.

Example 6.22. _item_selected

```
static void _item_selected(void* data, Evas_Object* o, const char* emmission,
                                                        const char* source) {
    printf("You clicked on the item with number %d\n", *((int *)data));
}
```

Finalmente el callback `_item_selected` será llamado cuando el usuario haga click con el botón del medio del ratón sobre un elemento del contenedor. Los datos contendrán el número para ese elemento en la rutina de creación mas arriba.

Ese es el final del código para la aplicación, despues viene la EDC requerida para que todo se muestre y funcione correctamente.

Example 6.23. La Edc

```
fonts {
    font: "Vera.ttf" "Vera";
}

collections {
    group {
        name, "container_ex";
        min, 300, 300;
        max, 800, 800;

        parts {
            part {
                name, "bg";
                type, RECT;
                mouse_events, 1;

                description {
                    state, "default" 0.0;
```

```
            color, 0 0 0 255;

            rel1 {
                relative, 0.0 0.1;
                offset, 0 0;
            }
            rel2 {
                relative, 1.0 1.0;
                offset, 0 0;
            }
        }
    }

    part {
        name, "header";
        type, RECT;
        mouse_events, 0;

        description {
            state, "default" 0.0;
            color, 255 255 255 255;

            rel1 {
                relative, 0.0 0.0;
                offset, 0 0;
            }

            rel2 {
                relative, 1.0 0.1;
                offset, 0 0;
            }
        }
    }

    part {
        name, "header_text_direction";
        type, TEXT;
        mouse_events, 0;

        description {
            state, "default" 0.0;
            color, 0 0 0 255;

            rel1 {
                relative, 0.0 0.0;
                offset, 0 10;
                to, "header";
            }
            rel2 {
                relative, 1.0 1.0;
                offset, 0 0;
                to, "header";
            }
            text {
                text, "direction";
                font, "Vera";
                size, 10;
            }
        }
    }

    part {
        name, "header_text_align";
        type, TEXT;
```

```
            mouse_events, 0;

            description {
                state, "default" 0.0;
                color, 0 0 0 255;

                rel1 {
                    relative, 0.0 0.0;
                    offset, 0 0;
                    to, "header_text_direction";
                }
                rel2 {
                    relative, 1.0 1.0;
                    offset, 110 0;
                    to, "header_text_direction";
                }
                text {
                    text, "align";
                    font, "Vera";
                    size, 10;
                }
            }
        }
```

Este archivo EDC espera tener la fuente Vera incorporada dentro de él, como es definido por la sección de fuentes al principio. Esto significa que cuando compiles la edc necesitas o bien la fuente Vera en el directorio actual, o bien dar a edje_cc la opción -fd y especificar el directorio a la fuente.

Después de definir las fuentes, las colecciones principales son definidas. La primera colección es la porción principal de la propia aplicación, el grupo "container_ex". Este grupo especifica la ventana principal de la aplicación. Como tal contiene las partes para el fondo, la cabecera, y el texto de la cabecera. Estas partes son todas bastante estándar con algúna (mínima) alineación hecha entre ellas.

Example 6.24. La parte del contenedor

```
        part {
            name, "container";
            type, RECT;
            mouse_events, 1;

            description {
                state, "default" 0.0;
                visible, 1;

                rel1 {
                    relative, 0.0 0.0;
                    offset, 0 0;
                    to, bg;
                }
                rel2 {
                    relative, 1.0 1.0;
                    offset, 0 0;
                    to, bg;
                }
                color, 0 0 0 0;
            }
        }
    }
```

```
        programs {
            program {
                name, "left_click";
                signal, "mouse,clicked,1";
                source, "container";
                action, SIGNAL_EMIT "left_click" "left_click";
            }

            program {
                name, "right_click";
                signal, "mouse,clicked,3";
                source, "container";
                action, SIGNAL_EMIT "right_click" "right_click";
            }
        }
    }
```

La parte del contenedor es entonces definida. La parte en sí es bastante simple, simplemente posicionada relativa al fondo e iniciada para recibir eventos de ratón. Después de definir las partes especificamos los programas para este grupo, de los cuales hay dos. El primer programa "left_click" especifica que va a pasar en caso de evento de click en el primer botón del ratón.

La acción es emitir una señal, los dos parámetros tras SIGNAL_EMIT encajan con los valores puestos en el callback en el código de la aplicación.

Hay un callback similar para el tercer botón del ratón como el primero, solo que emitiendo una señal ligeramente diferente.

Example 6.25. El grupo elemento

```
    group {
        name, "element";
        min, 80 18;
        max, 800 18;

        parts {
            part {
                name, "element.value";
                type, TEXT;
                mouse_events, 1;
                effect, NONE;

                description {
                    state, "default" 0.0;
                    visible, 1;

                    rel1 {
                        relative, 0.0 0.0;
                        offset, 0 0;
                    }
                    rel2 {
                        relative, 1.0 1.0;
                        offset, 0 0;
                    }
                    color, 255 255 255 255;

                    text {
                        text, "";
```

```
                        font, "Vera";
                        size, 10;
                    }
                }
            }
        }
        programs {
            program {
                name, "center_click";
                signal, "mouse,clicked,2";
                source, "element.value";
                action, SIGNAL_EMIT "item_selected" "item_selected";
            }
        }
    }
}
```

El grupo elemento especifica como ha de ser mostrado cada elemento del contenedor. Notarás que los nombres aquí dados encajan con los nombres que se buscan en el propio código de la aplicación mientras se crean los elementos.

En este grupo hay un programa que emitirá una señal "item_selected" cuando el botón del medio del ratón sea pulsado mientras nos encontramos sobre uno de los elementos en la lista.

Ese es el final del código EDC. Para compilar el código de la aplicación, un makefile semejante al de abajo podría ser usado.

Example 6.26. Makefile

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

container_ex: container/container_ex.c
        gcc -o container/container_ex container/container_ex.c $(CFLAGS) $(LIBS)
```

Y para crear el archivo EET, un simple 'edje_cc default.edc' debería ser suficiente si tienes el fichero Vera.ttf en el directorio actual.

Eso es todo, asumiendo todo va como planeado, deberías tener una aplicación simple en la cual pulsando los botones izquierdo/derecho del ratón mueve el contenedor a diferentes porciones de la ventana. Mientras que pulsar el botón del medio en elementos imprime el número del elemento.

Chapter 7. Epeg y Epsilon

En esta era moderna de fotografía digital la presentación se convierte en un problema debido al enorme volumen de imágenes que son creadas. Diferentemente a los viejos tiempos cuando el film se usaba con mesura ahora generamos cientos o miles de imágenes en una semana. La solución a este problema de presentación es el thumbnail, una imágen escalada en pequeño que puede ser indexada en una tabla o aplicación y rápidamente escaneada visualmente para encontrar las imágenes que desees. Pero el escalado de imagen es una operación muy intensiva, aunque podría tomar soko un segundo para tu potente Athlon el escalar una fotografía 1600x1200 a tu resolución requerida, si tienes 2000 fotografías eso tomará 30 minutos, y esto asume que no haces la operación manualmente en un editor como Photoshop o GIMP. El problema claramente pide una herramienta que pueda escalar imágenes con gran velocidad y eficiencia, con tanto control disponible como sea posible. La solución son dos librerías EFL : Epeg y Epsilon.

Epeg fué escrita por Raster para manejar exactamente el problema mencionado arriba con sus galerías de imágenes en rasterman.com. Es el thumbnailer más rápido del planeta. Con un API simple de usar, puede ser integrado en cualquier aplicación que quieras. El único inconveniente que tiene es que solo maneja JPEGs (de ahí su nombre), pero esto apenas es un problema considerando que todas las cámaras en el mercado usan JPEG como formato de salida por defecto.

Epsilon fué escrito por Atmos, inspirado por la velocidad de Epeg pero en respuesta a una necesidad de capacidad de thumbnailing multiformato. Epeg puede manejar JPEG, PNG, XCF, y GIF. Obviamente dado que no es una librería específica JPEG no maneja JPEG tan rápido como Epeg, pero puede usar a la propia Epeg para ganar las ventajas de velocidad que ésta provee. Epsilon, a diferencia de Epeg, es conforme al Thumbnail Managing Standard [http://triq.net/~jens/thumbnail-spec/index.html] de freedesktop.org. Como tal, extrae todos los thumbnails en la estructura de directorios especificada en el estándar (~/.thumbnails/) en lugar de a un lugar definido por el programador.

Ambas librerñias hacen tareas tan específicas que las API son muy simples de usar. Epeg tiene solo 17 funciones y Epsilon solo 9 haciendo estas librerías muy fáciles de aprender y usar en tus aplicaciones inmediatamente.

Receta: Thumbnailing simple con Epeg

Ben 'technikolor' Rockwood <benr@cuddletech.com>

La aplicaciñon mas simple de thumbnailing que pudieramos escribir tomaría solo dos argumentos, el nombre del archivo (imagen) de entrada y el nombre del archivo (thumbnail) de salida. El siguiente ejemplo de código usa Epeg exactamente para hacer eso.

Example 7.1. Thumbnail simple Epeg

```
#include <Epeg.h>

int main(int argc, char *argv[]){

        Epeg_Image * image;
        int w, h;

        if(argc < 2) {
                printf("Usage: %s input.jpg output.jpg\n", argv[0]);
                return(1);
        }

        image = epeg_file_open(argv[1]);
```

```
        epeg_size_get(image, &w, &h);
        printf("%s -  Width: %d, Height: %d\n", argv[1], w, h);
        printf("  Comment: %s", epeg_comment_get(image) );

        epeg_decode_size_set(image, 128, 96);
        epeg_file_output_set(image, argv[2]);
        epeg_encode(image);
        epeg_close(image);

        printf("... Done.\n");
        return(0);
}
```

Este ejemplo es bastante simplístico, no asegurándose de que el input es realmente un JPEG, pero adecuadamente muestra algunas características de la librería. Puede ser compilado de la siguiente manera:

Example 7.2.

```
gcc `epeg-config --libs --cflags` epeg-test.c -o epeg-test
```

La función `epeg_file_open` abre un JPEG para ser manipulado, devolviendo un puntero a Epeg_Image. Este puntero puede entonces ser pasado a otras funciones Epeg para manipularlo.

Dos funciones diferentes son usadas aquí para conseguir algo de información sobre la imagen de entrada: `epeg_size_get` y `epeg_comment_get`. Observa que ninguno de los valores retornados desde esas funciones es usado jamás en ninguna otra función Epeg, son simplemente para mostrar información. Un buen uso para esos valores podría ser inteligentemente definir el tamaño del thumbnail de salida, o modificar y pasar un comentario al thumbnail de salida.

El siguiente conjunto de funciones realmente hacen el trabajo. `epeg_decode_size_set` define el tamaño de salida del thumbnail. `epeg_file_output_set` define el nombre del archivo de salida. Y `epeg_encode` hace el peso pèsado del asunto. Observa que aunque no comprobamos el éxito aquí, `epeg_encode` devuelve un int permitiéndonos comprobarlo.

Una vez el thumbnail ha sido creado, símplemente llama a `epeg_close` para terminar el asunto.

Aunque este ejemplo es quizás demasiado simplista puedes ver como lo básico funciona. Epeg tiene también funciones para reducción, comentado del thumbnail, habilitar/deshabilitar los comentarios, conversión de espacio de color, y cambios de configuración de calidad que pueden ser usados para conseguir cualquier resultado que quieras.

Receta: Thumbnailing simple con Epsilon

Ben 'technikolor' Rockwood <benr@cuddletech.com>

Epsilon crea thumbnails que son conformes al Thumbnail Managing Standard [http://triq.net/~jens/thumbnail-spec/index.html] de freedesktop.org. Thumbnails pueden ser creados para una variedad de formatos, incluyendo soporte nativo PNG , soporte Epeg, o cualquier formato soportado por Imlib2. Miremos una simple aplicación Epsilon similar al ejemplo Epeg anterior.

Example 7.3. Thumbnail simple Epsilon

```
#include <stdio.h>
#include <Epsilon.h>

int main(int argc, char *argv[]){

        Epsilon * image = NULL;
        Epsilon_Info *info;

        if(argc < 1) {
                printf("Usage: %s input_image\n", argv[0]);
                return(1);
        }

        epsilon_init();

        image = epsilon_new(argv[1]);

        info = epsilon_info_get(image);
        printf("%s -  Width: %d, Height: %d\n", argv[1], info->w, info->h);


        if (epsilon_generate(image) == EPSILON_OK) {
                printf("Thumbnail created!\n");
        } else {
                printf("Generation failed\n");
        }
        epsilon_free(image);

        return(0);
}
```

Puede ser compilado de la siguiente manera:

Example 7.4.

```
gcc `epsilon-config --libs --cflags` epsilon-simple.c -o epsilon-simple
```

Observarás casi inmediatamente que no se acepta ningún nombre de archivo de entrada, ni se usa ninguna función de salida. El Thumbnail Managing Standard especifica que todos los thumbnails han de ser creados en el arbol de directorios ~/.thumbnail. Este almacén central de thumbnails permite compartir thumbnails entre multiple applicaciones que se adhieren al estándar. Tras compilar y ejecutar el código de ejemplo con una imagen busca la imagen en ~/.thumbnails/large. Los thumbnails también se nombran de acuerdo con el estándar, reemplazando el nombre original con una suma de comprobación MD5 de manera que incluso si la imagen cambia de nombre el thumbnail no necesita ser regenerado.

En nuestro ejemplo empezamos verificando que obtenemos una imagen de entrada para hacer un thumbnail y entonces inicializamos Epsilon usando `epsilon_init` function. `epsilon_new` accepta un único argumento, la imagen de la que hacer el thumbnail, y devuelve un puntero a epsilon que es usado por otras funciones.

Epsilon tiene la habilidad de obtener algo de información básica de tus imágenes. En el ejemplo de arriba usamos `epsilon_info_get` para devolver una estructura Epsilon_Info conteniendo ek tiempo de modificación de la imagen de entrada (mtime), lugar (URI), anchura, altura, y tipo MIME. Aquí simplemente damos la anchura y altura de la imagen usando los elementos `w` y `h` de la estructura info.

`epsilon_generate` es el peso pesado. Esta función generará el thumbnail y lo colocará en el lugar apropiado. Su valor de retorno indica el éxito (o no), para el cual la cabecera Epsilon provee definiciones de macro CPP: `EPSILON_FAIL` y `EPSILON_OK`.

La limpieza se efectúa mediante `epsilon_free`.

Epsilon, como aquí se ha visto, es muy simple de usar e integrar en cualquier aplicación que se apoye en thumbnails. No solo se provee un simple API sino también integración con el estándar reinante de thumbnailing sin costo extra. âra información adicional sobre Epsilon, mira los documentos Doxygen de Epsilon en Enlightenment.org.

Chapter 8. Etox

Etox es una librerÍa avanzada de composición y maquetado de texto basada en Evas, que permite funcionalidad mayor y mas allá de la provista por Evas. Es capaz de simplificar como mostrar, mover, redimensionar, poner en capas, y recortar el texto, asÍ como alinear texto, ajustar, y modificar. Etox puede incluso inteligentemente ajustar obstáculos y aplicar *styles* predefinidos. Casi cualquier aspecto de manejo de texto puede ser manejado fácil y eficientemente con Etox.

Receta: Perspectiva general de Etox

Ben 'technikolor' Rockwood <benr@cuddletech.com>

Para empezar a usar Etox rápidamente, es útil un ejemplo simple. En el siguiente código de ejemplo crearemos un Evas X11 usando Ecore_Evas y entonces pondremos algo de texto Etox en él.

Example 8.1. ejemplo Etox

```
#include <Ecore_Evas.h>
#include <Ecore.h>
#include <Etox.h>

#define WIDTH 400
#define HEIGHT 200

        Ecore_Evas  *   ee;
        Evas        *   evas;
        Evas_Object *   base_rect;
        Evas_Object *   etox;
        Etox_Context *  context;

int main(){

        ecore_init();

   ee = ecore_evas_software_x11_new(NULL, 0,  0, 0, WIDTH, HEIGHT);
        ecore_evas_title_set(ee, "ETOX Test");
        ecore_evas_borderless_set(ee, 0);
        ecore_evas_show(ee);

   evas = ecore_evas_get(ee);
        evas_font_path_append(evas, ".");

   base_rect = evas_object_rectangle_add(evas);
        evas_object_resize(base_rect, (double)WIDTH, (double)HEIGHT);
        evas_object_color_set(base_rect, 255, 255, 255, 255);
        evas_object_show(base_rect);

   etox = etox_new(evas);
        evas_object_resize(etox, WIDTH, HEIGHT);


   context = etox_get_context(etox);
        etox_context_set_color(context, 0, 0, 0, 255);
        etox_context_set_font(context, "Vera", 32);
        etox_context_set_align(context, ETOX_ALIGN_LEFT);

        etox_set_soft_wrap(etox, 1);
        etox_set_text(etox, "Welcome to the world of Etox!");
```

```
        evas_object_show(etox);

        ecore_main_loop_begin();

        return 0;
}
```

Este ejemplo puede compilarse de la siguiente manera:

Example 8.2.

```
gcc `etox-config --libs --cflags` `ecore-config --libs --cflags` etox-test.c -o et
```

La mayoría de este ejemplo son funciones estándar Ecore_Evas así que nos concentraremos en las partes relacionadas con Etox. Observa que usamos la función Evas`evas_font_path_append()` para definir nuestro path de fuentes, esto es algo que Etox no hará por tí.

Tu texto Etox siempre empezará añadiendo un nuevo Etox usando la función `etox_new()` que devuelve un Evas_Object. Como tu Etox es un objeto Evas puede ser manipulado como tal. Las funciones de ajuste como recortar y ajustar son dependientes del tamaño del propio Etox, por lo tanto `evas_object_resize()` necesita ser llamada para definir el tamaño apropiado del Etox. Observa que el area del objeto *no* será igual por defecto al tamaño del propio Evas.

Etox usa el concepto de contextos. Un `context` es un conjunto de parámetros como color, fuente, alineación, estilo, y marcas que son aplicados a un cierto conjunto de texto. Cada objeto Etox tiene al menos un contexto asociado con él que es creado cuando se llama a `etox_new()`. Por esta razón la función `etox_context_new()` function *solo* necesita ser llamada cuando se crean contextos adicionales.

Una vez has usado `etox_new()` para añadir tu objeto Etox necesitas usar `etox_get_context()` para devolver un puntero a `Etox_Context` que puede entonces ser pasado a otras funciones de contexto para modificar los atributos de tu texto. En el ejemplo cambiamos el color, fuente y alineación de nuestro contexto.

Dos de las carácteristicas mas interesantes y simplisticas de Etox son su habilidad de inteligentemente ajustar texto y de interpretar un carácter de newline estándar de C (\n) como ajuste. Esas son características que el propio Evas no proporciona, es responsabilidad del programador asegurar que el texto no se sale del canvas..

El ajustado inteligente viene en dos formas que no son mutuamente exclusivas. La primera es el ajustado suave, que ajustará el texto cuando un carácter vaya a exceder la anchura del canvas. La segunda es ajuste de palabra, que ajustará texto cuando una palabra vaya a exceder la anchura del canvas. Tipicamente la segunda manera es deseable de manera que obtengamos "Esta es (ajuste) mi cadena" en lugar de "Esta es m(ajuste)i cadena". Observa, sin embargo, que el ajuste de palabra no funcionará a menos que el ajuste suave haya sido ya habilitado, por lo tanto el ajuste de palabra requiere llamar a *ambas* funciones, `etox_set_soft_wrap()` y `etox_set_word_wrap()` functions.

Una nota final sobre el ajuste es que por defecto el ajuste insertará un `wrap marker` en tu cadena de salida, un signo "+" por defecto. Esta marca indica que un ajuste ha ocurrido y es impresa como primer carácter en la siguiente linea. Tu cadena por lo tanto será mas bien como esto: "Esta es mi (ajuste) +cadena". Si prefieres que Etox ajuste silenciosamente sin marca, simplemente configura la marca a ser nada usando la función `etox_context_set_wrap_marker()`.

Las cadenas de texto Etox se configuran usando `etox_set_text()`. Es importante observar que la cadena se aplica al propio Etox y no al contexto. No hay asociación directa entre la cadena y el contexto, que facilita facil modificación del mostrado del texto sin tener que cambiar el contexto, o viceversa.

Aunque este es un ejemplo muy simple del uso de Etox, mucho más puede ser hecho y como puedes ver el API es simple y limpio, rellenando muchas de las necesidades de manejo de texto que Evas no provee.

Chapter 9. Edje

Edje es una compleja librería de diseño gráfico y maquetación. Su propósito es abstraer todo elemento del interfaz de tu aplicación Evas del propio código.

Una aplicación Edje consiste en dos partes: El código C que forma tu aplicación y una colección de objetos edje (EDC) que describe cada elemento de tu interfaz. Ambas están conectadas por señales que son emitidas desde tu EDC y recibidas por callbacks en el código de tu aplicación. Usando este modelo de señales el código de tu aplicación está completamente desinteresado en que aspecto tiene tu interfaz, mientras reciba una señal. Y como las señales son procesadas por callbacks, no hay requerimiento de que tu interfaz envie cada señal disponible, haciendo posible aplicaciones en gran escala y aplicaciones de tamaño "demo" con un único binario. Tanto si tu interfaz usa botones como si usa una drag-bar para enviar datos, a tu aplicación no le importa.

Receta: Una plantilla para crear aplicaciones Edje

Ben 'technikolor' Rockwood <benr@cuddletech.com>

El siguiente ejemplo es una plantilla que puede ser usada para iniciar rápida y fácilmente una aplicación Edja. Se parece a la plantilla encontrada en el capítulo Evas, dado que esta tambien usa Ecore_Evas.

Example 9.1. Plantilla Edje

```
#include <Ecore_Evas.h>
#include <Ecore.h>
#include <Edje.h>

#define WIDTH 100
#define HEIGHT 100

int app_signal_exit(void *data, int type, void *event);

        /* GLOBALS */
        Ecore_Evas  *   ee;
        Evas        *   evas;
        Evas_Object *   edje;

        Evas_Coord   edje_w, edje_h;


int main(int argv, char *argc[]){

        ecore_init();
        ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, app_signal_exit, NULL);

        ecore_evas_init();

   ee = ecore_evas_software_x11_new(NULL, 0,  0, 0, WIDTH, HEIGHT);
        ecore_evas_title_set(ee, "TITLE");
        ecore_evas_borderless_set(ee, 0);
        ecore_evas_shaped_set(ee, 0);
        ecore_evas_show(ee);


   evas = ecore_evas_get(ee);
```

```
        evas_font_path_append(evas, "edje/fonts/");

        edje_init();
   edje = edje_object_add(evas);
        edje_object_file_set(edje, "edje/XXX.eet", "XXX");
        evas_object_move(edje, 0, 0);
        edje_object_size_min_get(edje, &edje_w, &edje_h);
        evas_object_resize(edje, edje_w, edje_h);
        evas_object_show(edje);

        ecore_evas_resize(ee, (int)edje_w, (int)edje_h);
        ecore_evas_show(ee);

        /* Insert Objects and callbacks here */

        ecore_main_loop_begin();

        return 0;
}


int app_signal_exit(void *data, int type, void *event){

        printf("DEBUG: Exit called, shutting down\n");
        ecore_main_loop_quit();
        return 1;
}
```

Compilar esta plantilla de la siguiente manera:

```
 gcc `edje-config --cflags --libs` `ecore-config --cflags --libs` edje_app.c -o ed
```

Las llamadas importantes a las que mirrar estan contenidas en el bloque Edje, siguiendo a `edje_init()`.

`edje_object_file_set()` define que EET Edje es usada así como el nombre de la colección a usar.

El resto de funciones Edje/Evas son necesitadas para redimensionar la ventana X11 que acomoda tu Edje. Empezamos moviendo la ventana Evas y adquiriendo el tamaño mínimo del propio Edje usando `edje_object_size_min_get()`. Usando entonces `evas_object_resize()` podemos redimensionar el Edje, el cual es en realidad un objeto Evas, al tamaño del propio Evas. Tras esto podemos mostrar el Edje y entonces redimensionar el propio Evas (y gracias a Ecore la ventana tambien) usando `ecore_evas_resize()`.

Mas allá de esto callbacks pueden ser vinculadas a tu interfaz.

Chapter 10. Edje EDC y Embryo

Los archivos fuente de colecciones de datos Edje (EDC) permiten una fácil creación de ricos y potentes interfaces gráficos. Tu aplicación Edje está dividida en dos partes distintas, el código de aplicación (usando llamadas de `Edje.h`) y la descripción de interfaz en el EDC. La única conectividad requerida entre tu interfaz y el código de tu aplicación son las señales emitidas por tu interfaz y son recibidas por callbacks Edje en el código de tu aplicación.

Un EDC está compuesto de varias secciones principales describiendo las imágenes y fuentes que son usadas en el interfaz, descripciones de como las varias partes `part` del interfaz son maquetadas, y descripciones de acciones `program` que ocurren cuando se interactúa con tu interfaz. Esta funcionalidad puede ser suplementada usando el lenguage de scripting Embryo para añadir programabilidad de estilo C en la propia EDC Edje.

El resultado final de una EDC, incluyendo todas sus fuentes e imágenes, es un único EET. Dado que el interfaz completo es disponible en un único archivo distribución de "temas" es drásticamente simplificada.

Aunque las EDC Edje podrían ser consideradas "temas" son mucho más. Un "tema" tradicional es un fichero o grupo de ficheros que aumentan algún interfaz gráfico existente cmbiando el color de los elementos o reemplazando las imágenes que componen el propio interfaz. Pero esos métodos son insuficientes para cambiar realmente el diseño del interfaz de una aplicación, limitando a los diseñadores de temas de modificarlos y a menudo requiriendo un rediseño de la aplicación en algún momento para expandir las capacidades del interfaz para una mayor funcionalidad. Una aplicación GTK siempre tendrá el mismo aspecto, a persar del tema que use. Un simple ejemplo es que una aplicación GTK o QT siempre tendrá una forma rectangular y si tiene un borde no puedes quitarle el borde con un tema. sin embargo , una aplicación Edje podría cambiar de borde rectangular a ovalado con una simple modificación de la EDC, o podría eliminar y recolocar todos los elementos del interfaz sin tocar jamás el código de la aplicación. De esta manera Edje permite un control y una flexibilidad mayos que la provista por cualquier otra solución en la comunidad Open Source y permite el modelo de programación abierta para permitir incluso a los no-programadores (como muchos diseñadores de temas son) que contribuyan y modifiquen cosas como consideren oportuno.

Receta: conmutador Edje/Embryo

Corey 'atmos' Donohoe <atmos@atmos.org>

Hace un largo tiempo Raster [http://www.rasterman.com] hizo Edje, y era bueno. Los cavernícloas que descubrieron Edje en las paredes de las cavernas (#edevelop) estaban maravillados, pero pronto habían muchas desventajas. Dada la suficiente creatividad podías hacer cosas, conmutadores por ejemplo, pero era alquimia hacerlo apropiadamente. Para propósito histórico, se provee un conmutador Edje sin Embryo , tan retorcido como es. Mira el ejemplo Edje without Embryo mas abajo.

Notarás que tienes que hablar en señales a tu aplicación para determinar el estado de tu conmutador. Así que sin mas preámbulo, aqui hay un conmutador Edje usando Embryo, de una manera *mucho* mas elegante.

El scripting Embryo dentro de Edje, en lo sucesivo scripting EE, te da variables. Puedes tener enteros, números en punto flotante, y cadenas. Esto significa basicamente que puedes tener algo de lógica de programación en tus edjes. Nada comlejo, como estructuras compejas, pero variables simples contenidas en un grupo podrían asemejarse a los miembros de estructuras.

La primera parte de EE es escoger tus variables. En este simple ejemplo solo tenemos una variable, y la involucras en un grupo edje declarando un bloque *script { ... }*. *button_toggle_state* es implicitamente un entero, y será usado como variable booleana para permitirnos saber si el botón de conmutación está activado o desactivado. Lo bueno de esta variable es que la podemos usar como una manera de comunicación entre nuestra aplicación y nuestro edje. Además puedes estar tranquilo sabiendo (asum-

iendo que lo hiciste correctamente) que ninguna artimaña de edje va a lanzar tu aplicación al limbo.

Example 10.1. Creando la variable

```
collections {
    group {
        name: "Toggler";
        script {
            public button_toggle_state;
        }
        parts {
            part {
                ...
            }
        }
        programs {
            program {
                ...
            }
        }
    }
}
```

La segunda parte del scripting EE es inicializar las variables. En su mayor parte puedes asumir que estas variables se inicializarán a cero, pero es buena costumbre iniciarlas tú mismo. Edje emite una señal "load" cuando el grupo es cargado en memoria, esta es tu oportunidad para iniciar tus variables embryo.

Example 10.2. Inicializando variables

```
program {
    name: "group_loaded";
    signal: "load";
    source: "";
    script {
        set_int(button_toggle_state, 0);
    }
}
```

La tercera parte es propiamente dar un aspecto a tu edje. Para este ejemplo se usan rectángulos, pero imágenes e incluso texto deberían tambien funcionar correctamente. Hay un objeto de background simplemente por consistencia, y hay un rectángulo llamado "toggler". toggler tiene dos estados, el estado por defecto (implicitamente deshabilitado) y habilitado. Cuando toggler es pulsado debería (lo adivinaste) cambiar al otro estado. off -> on, on -> off. toggler va a tener su estado por defecto (deshabilitado) de color rojo, y su estado habilitado azul de manera que puedan ser fácilmente diferenciados. El background va a ser blanco porque no es rojo o azul :D

Example 10.3. El botón conmutador

```
collections {
    group {
        name: "Toggler";
        script {
            public button_toggle_state;
        }
        parts {
            part {
                name: "background";
                type: RECT;
                mouse_events: 0;
                description {
                    state: "default" 0.0;
                    color: 255 255 255 255;
                    rel1 { relative: 0.0 0.0; offset: 0 0; }
                    rel2 { relative: 1.0 1.0; offset: 0 0; }
                }
            }
            part {
                name: "toggle";
                type: RECT;
                mouse_events: 1;
                description {
                    state: "default" 0.0;
                    color: 255 0 0 255;
                    rel1 { relative: 0.0 0.0; offset: 10 10; }
                    rel2 { relative: 1.0 1.0; offset: -10 -10; }
                }
                description {
                    state: "on" 0.0;
                    color: 0 0 255 255;
                    rel1 { relative: 0.0 0.0; offset: 10 10; }
                    rel2 { relative: 1.0 1.0; offset: -10 -10; }
                }
            }
        }
        programs {
            program {
                name: "group_loaded";
                signal: "load";
                source: "";
                script {
                    set_int(button_toggle_state, 0);
                }
            }
        }
    }
}
```

La cuarta parte está conectándose en los eventos de ratón para provocar la conmutación como programas edje. No solo cambiando la variable Embryo, sino tambien la apariencia de nuestro edje. Este ejemplo usa pulsación normal de botón izquierdo para cambiar el estado del conmutador, en términos de edje "mouse,clicked,1". Este ejemplo no usa una llamada a la función incorporada en Embryo *set_state* , sino que emite señales que son recibidas por otros programas. El razonamiento tras esta aproximación es permitir transiciones visuales suaves entre los dos estados. La llamada a la función Embryo set?state es un cambio de estado inmediato, y no tiene un aspecto tan agradable como la transición SINUSOIDAL usada en los siguientes fragmentos.

Example 10.4. Hooking into the mouse events

```
collections {
    group {
        name: "Toggler";
        script {
            public button_toggle_state;
        }
        parts {
            part {
                 ...
            }
        }
        programs {
            program {
                name: "toggle_icon_mouse_clicked";
                signal: "mouse,clicked,1";
                source: "toggle";
                script {
                    if(get_int(button_toggle_state) == 0) {
                        set_int(button_toggle_state, 1);
                        emit("toggle,on", "");
                    }
                    else {
                        set_int(button_toggle_state, 0);
                        emit("toggle,off", "");
                    }
                }
            }
            program {
                name: "toggle_on";
                signal: "toggle,on";
                source: "";
                action: STATE_SET "on" 0.0;
                target: "toggle";
                transition: SINUSOIDAL 0.5;
            }
            program {
                name: "toggle_off";
                signal: "toggle,off";
                source: "";
                action: STATE_SET "default" 0.0;
                target: "toggle";
                transition: SINUSOIDAL 0.5;
             }
         }
     }
 }
```

La quinta parte es sopesar el escenario presentado. Esta es solo la punta del iceberg con respecto al scripting EE. Puedes añadir muchas mas variables para llevar la pista de estados internos que no están relacionados en absoluto con tus aplicaciones. Hay matizaciones entre esto y el uso práctico de las variables Embryo, sin embargo entender estos bloques básicos hará mucho mas simple el escribir o trabajar con aplicaciones scriptadas en EE mucho mas simple.

- ¿Que hay de malo en la técnica aquí presentada?
- ¿Que pasa si la aplicación quiere el conmutador "on" por defecto?

Puedes usar una script similar a la siguiente para construir este ejemplo.

Example 10.5. Build script

```
#!/bin/sh -e
THEME="default"
APPNAME=""
edje_cc -v $THEME.edc $THEME.eet
if [ $? = "0" ]; then
    if [ "$APPNAME" = "" ]; then
        echo "Build was successful"
    else
        PREFIX=`dirname \`which $APPNAME\` | sed 's/bin//'`
        sudo cp $THEME.eet $PREFIX"share/$APPNAME/themes/"
        echo -n "Installed theme to "
        echo $PREFIX"share/$APPNAME/themes/"
    fi
else
    echo "Building failed"
fi
```

Example 10.6. Edje toggle without Embryo

```
images { }

collections {
    group {
        name, "Rephorm";
        min, 50 50;
        max, 75 75;
        parts {
            part {
                name, "Clip";
                type, RECT;
                mouse_events, 0;
                description {
                    state, "default" 0.0;
                    visible, 1;
                    rel1 { relative, 0.0 0.0; offset, 5 5; }
                    rel2 { relative, 1.0 1.0; offset, -5 -5; }
                    color, 255 255 255 255;
                }
                description {
                    state, "hidden" 0.0;
                    visible, 1;
                    rel1 { relative, 0.0 0.0; offset, 5 5; }
                    rel2 { relative, 1.0 1.0; offset, -5 -5; }
                    color, 255 255 255 128;
                }
            }
            part {
                name, "On";
                type, RECT;
                mouse_events, 1;
                clip_to, "Clip";
                description {
```

```
                state, "default" 0.0;
                visible, 0;
                rel1 { relative, 0.0 0.0; offset, 5 5; }
                rel2 { relative, 1.0 1.0; offset, -5 -5; }
                color, 255 0 0 0;
            }
            description {
                state, "visible" 0.0;
                visible, 1;
                rel1 { relative, 0.0 0.0; offset, 5 5; }
                rel2 { relative, 1.0 1.0; offset, -5 -5; }
                color, 255 0 0 255;
            }
        }
        part {
            name, "Off";
            type, RECT;
            mouse_events, 1;
            clip_to, "Clip";
            description {
                state, "default" 0.0;
                visible, 1;
                rel1 { relative, 0.0 0.0; offset, 5 5; }
                rel2 { relative, 1.0 1.0; offset, -5 -5; }
                color, 0 0 255 255;
            }
            description {
                state, "visible" 0.0;
                visible, 0;
                rel1 { relative, 0.0 0.0; offset, 5 5; }
                rel2 { relative, 1.0 1.0; offset, -5 -5; }
                color, 0 0 255 0;
            }
        }
        part {
            name, "Grabber";
            type, RECT;
            mouse_events, 1;
            repeat_events, 1;
            clip_to, "Clip";
            description {
                state, "default" 0.0;
                visible, 1;
                rel1 { relative, 0.0 0.0; offset, 5 5; }
                rel2 { relative, 1.0 1.0; offset, -5 -5; }
                color, 255 255 255 0;
            }
        }
    }
    programs {
        program {
            name, "ToggleOn";
            signal, "mouse,clicked,1";
            source, "Off";
            action, STATE_SET "visible" 0.0;
            target, "Off";
            target, "On";
            transition, SINUSOIDAL 0.5;
        }
        program {
            name, "ToggleOff";
            signal, "mouse,clicked,1";
            source, "On";
            action, STATE_SET "default" 0.0;
```

```
                target, "Off";
                target, "On";
                transition, SINUSOIDAL 0.5;
            }
            program {
                name, "GrabberIn";
                signal, "mouse,in";
                source, "Grabber";
                action, STATE_SET "default" 0.0;
                target, "Clip";
                transition, SINUSOIDAL 0.5;
            }
            program {
                    name, "GrabberOut";
                signal, "mouse,out";
                source, "Grabber";
                action, STATE_SET "hidden" 0.0;
                target, "Clip";
                transition, SINUSOIDAL 0.5;
            }
        }
    }
}
```

Chapter 11. EWL

La Enlightened Widget Library (EWL) es un conjunto de herramientas de widget que construye sobre las fundaciones construidas por las otras librerías en las EFL. Ewl usa Evas como backend de renderizado y su apariencia es abstraída para ser manejada por Edje.

EWL es similar a muchos de los otros toolkits que hay incluyendo GTK, QT o MOTIF. Las APIs son distintas pero los conceptos son los mismos.

Receta: Introducción a EWL

dan 'dj2' sinclair <zero@perplexity.org>

Mediante el uso de la Enlightened Widget Library (EWL), un montón de potencia puede ser puesta en manos de los programadores con poco esfuerzo.

Esta introducción a EWL mostrará comocrear una simple aplicación para ver texto con una barra de menú y un diálogo de archivo. El area de texto tendrá barras de desplazamiento y permitirá tambien desplazarse usando las teclas del teclado, o la rueda del ratón.

Example 11.1. Includes y declaraciones

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <Ewl.h>

#define PROG    "EWL Text Viewer"

/* globals */
static Ewl_Widget *main_win = NULL;
static Ewl_Widget *fd_win = NULL;

/* pre-declarations */
static void destroy_cb(Ewl_Widget *, void *, void *);
static void destroy_filedialog_cb(Ewl_Widget *, void *, void *);
static void open_file_cb(Ewl_Widget *, void *, void *);
static void home_cb(Ewl_Widget *win, void *ev, void *data);
static void file_menu_open_cb(Ewl_Widget *, void *, void *);
static void key_up_cb(Ewl_Widget *, void *, void *);

static char *read_file(char *);
static void mk_gui(void);
```

El único include requerido para escribir una aplicación Ewl es la declaración <Ewl.h>. Hacemos la ventana principal y la de diálogo de archivo globales para facilitar un acceso mas facil en las funciones de callback. Estas no necesitan ser globales, pero para los propósitos de este ejemplo, es mas simple si lo son.

Example 11.2. main

```
/* lets go */
int main(int argc, char ** argv) {
    ewl_init(&argc, argv);
    mk_gui();
    ewl_main();
    return 0;
}
```

La función principal para nuestro visor de texto es muy simple. Empezamos inicializando ewl mediante `ewl_init()` call. Ewl toma las entradas argc y argv para hacer algún parseado propio de la linea de comandos. Esto incluye cosas como iniciar el tema a usar (--ewl-theme) o iniciar el motor de renderizado (--ewl-software-x11, --ewl-gl-x11, etc.).

`ewl_init()` se encarga de todo el trabajo sucio de inicializar las otras librerías requeridas, abstrayendo todo eso del programador en un interfaz simple.

La llamada a mk_gui iniciará la ventana principal y cualquier contenido requerido.

La llamada a `ewl_main()` inicia el ciclo principal de proceso, y cuando sale maneja toda la finalización requerida por la aplicación, por lo tanto no hay llamada de finalización desde la rutina principal.

Example 11.3. mk_gui: creando la ventana

```
/* build the main gui */
static void mk_gui(void) {
    Ewl_Widget *box = NULL, *menu_bar = NULL;
    Ewl_Widget *text_area = NULL, *scroll = NULL;

    /* create the main window */
    main_win = ewl_window_new();
    ewl_window_title_set(EWL_WINDOW(main_win), PROG);
    ewl_window_name_set(EWL_WINDOW(main_win), PROG);
    ewl_window_class_set(EWL_WINDOW(main_win), PROG);

    ewl_object_size_request(EWL_OBJECT(main_win), 200, 300);
    ewl_object_fill_policy_set(EWL_OBJECT(main_win), EWL_FLAG_FILL_FILL);

    ewl_callback_append(main_win, EWL_CALLBACK_DELETE_WINDOW, destroy_cb, NULL);
    ewl_widget_show(main_win);
```

La primera cosa que necesitamos hacer para arrancar nuestra aplicación es crear la ventana principal de la aplicación. Esto es hecho mediante la llamada a `ewl_window_new()`. Una vez tenemos la ventana podemos continuar iniciando el título (tal como aparecerá en la barra del gestor de ventanas sobre la aplicación), el nombre y la clase de la ventana.

Una vez la información por defecto es puesta en la ventana requerimos un tamaño por defecto para la ventana de 200x300 por medio de la llamada a `ewl_object_size_request`. Junto con el tamaño por defecto podríamos haber puesto un tamaño mínimo y máximo por medio de llamadas a `ewl_object_minimum_size_set` y `ewl_object_maximum_size_set`. Pero como esto no es requerido para esta aplicación se dejan fuera.

La configuración final de la ventana ses hecha seleccionando la política de relleno con

`ewl_object_fill_policy_set`. Esto selecciona como Ewl empaquetará los widgets en la ventana, con unos valores posibles de:

EWL_FLAG_FILL_NONE	No rellenar o encoger en ninguna dirección
EWL_FLAG_FILL_HSHRINK	Encoger horizontalmente
EWL_FLAG_FILL_VSHRINK	Encoger verticalmente
EWL_FLAG_FILL_SHRINK	Encoger tanto horizontál como verticálmente
EWL_FLAG_FILL_HFILL	Rellenar horizontalmente
EWL_FLAG_FILL_VFILL	Rellenar verticalmente
EWL_FLAG_FILL_FILL	Rellenar tanto horizontál como verticalmente
EWL_FLAG_FILL_ALL	Encoger y Rellenar a la vez

Despues de definir las propiedades de la ventana se añade un callback para capturar la destrucción de la ventana principal con `ewl_callback_append()`. La función `destroy_cb()` será llamada si alguien requiere que la ventana sea destruida de alguna manera.

Enseñamos la ventana principal con una llamada a `ewl_widget_show()`. Si la función `ewl_widget_show()` no es llamada nada aparecerá en la pantalla. Todos los widgets están ocultos hasta que son mostrados explicitamente. Lo opuesto a esto es `ewl_widget_hide()` que eliminará a un widget de la pantalla.

Example 11.4. El contenedor principal

```
/* create the main container */
box = ewl_vbox_new();
ewl_container_child_append(EWL_CONTAINER(main_win), box);
ewl_object_fill_policy_set(EWL_OBJECT(box), EWL_FLAG_FILL_FILL);
ewl_widget_show(box);
```

Podríamos empaquetar todos nuestros widgets en la propia ventana principal, pero eso podría causar problemas mas tarde si quisieramos cambiar las cosas fácilmente, así que en su lugar crearemos una caja dentro de la ventana principal para que mantenga todos nuestros widgets.

Esto es hecho creando una caja vertical con `ewl_vbox_new()`. Se toma entonces la caja y se añade a la lista de hijos de la ventana con `ewl_container_child_append()`. Después de añadir a la ventana seleccionamos una política de relleno para rellenar tanto en horizontal como en vertical con `ewl_object_fill_policy_set`, y mostramos el widget con `ewl_widget_show()`.

El orden en el que pones los widgets en el container afectará la manera en que se muestre la aplicación. El primer widget empaquetado será el primer widget en ser mostrado. Dado que especificamos una caja vertical empezaremos empaquetando los widgets desde arriba hacia abajo.

Example 11.5. Crear la barra de menú

```
/* create the menu bar */
menu_bar = ewl_hbox_new();
ewl_container_child_append(EWL_CONTAINER(box), menu_bar);
ewl_object_fill_policy_set(EWL_OBJECT(menu_bar), EWL_FLAG_FILL_HSHRINK);
ewl_object_alignment_set(EWL_OBJECT(menu_bar), EWL_FLAG_ALIGN_LEFT);
ewl_box_set_spacing(EWL_BOX(menu_bar), 4);
ewl_object_padding_set(EWL_OBJECT(menu_bar), 5, 5, 5, 5);
ewl_widget_show(menu_bar);
```

El primer widget que ponemos en el sitio es la barra de menú. Pondremos los propios contenidos de la barra de menú mas tarde después de que otros widgets hayan sido creados pero necesitamos poner la barra de menú en su sitio primero.

Las llamadas son las mismas que muchas que has visto antes, añadirnos a la ventana madre, iniciar la política de relleno, enseñar el widget. Las que no se han visto antes incluyen `ewl_object_alignment_set()`, esto seleccionará como está alineado el widget dentro de su contenedor. En este caso estamos usando EWL_FLAG_ALIGN_LEFT, pero podríamos haber usado alguna de las otras alineaciones disponibles incluyendo:

- EWL_FLAG_ALIGN_CENTER
- EWL_FLAG_ALIGN_LEFT
- EWL_FLAG_ALIGN_RIGHT
- EWL_FLAG_ALIGN_TOP
- EWL_FLAG_ALIGN_BOTTOM

Así que el menú estará alineado con el lado izquierdo de la caja principal.

Entonces especificamos el espaciado de elementos dentro de la caja del menú. Esto dará un poco más de espacio entre nuestros elementos de menú y se hace con `ewl_box_set_spacing()`. Despues de cambiar el espacio cambiamos el relleno alrededor de la caja como un todo con la llamada a `ewl_object_padding_set()`, esto incrementará la cantidad de espacio que queda alrededor del objeto como un todo.

Example 11.6. Crear el panel de desplazamiento

```
/* create the scrollpane */
scroll = ewl_scrollpane_new();
ewl_container_child_append(EWL_CONTAINER(box), scroll);
ewl_object_fill_policy_set(EWL_OBJECT(scroll), EWL_FLAG_FILL_FILL);
ewl_scrollpane_hscrollbar_flag_set(EWL_SCROLLPANE(scroll),
                                   EWL_SCROLLBAR_FLAG_AUTO_VISIBLE);
ewl_scrollpane_vscrollbar_flag_set(EWL_SCROLLPANE(scroll),
                                   EWL_SCROLLBAR_FLAG_AUTO_VISIBLE);
ewl_widget_show(scroll);
```

El panel de desplazamiento va a ser la ventana madre de nuestro objeto de texto. El panel de desplazamiento nos provee con toda la magia para controlar las barras de desplazamiento y el propio desplazamiento.

El panel de desplazamiento es creado con una llamada a `ewl_scrollpane_new()`, y entonces procedemos a añadir el panel de desplazamiento a la caja principal, y seleccionar su política de relleno.

Las llamadas a ewl_scrollpane_[hv]scrollbar_flag_set() le dicen a Ewl como deberían comportarse las barras de desplazamiento. Los valores posibles son:

- EWL_SCROLLBAR_FLAG_NONE
- EWL_SCROLLBAR_FLAG_AUTO_VISIBLE
- EWL_SCROLLBAR_FLAG_ALWAYS_HIDDEN

Una vez las barras de desplazamiento han sido iniciadas le decimos a Ewl que muestre el widget.

Example 11.7. Crear el area de texto

```
/* create the text area */
text_area = ewl_text_new("");
ewl_container_child_append(EWL_CONTAINER(scroll), text_area);
ewl_object_padding_set(EWL_OBJECT(text_area), 1, 1, 1, 1);
ewl_widget_show(text_area);
```

El area de texto será responsable de mantener el texto que mostramos en nuestro visor. El widget es creado con una simple llamada a `ewl_text_new()`. Esto causará la creación del área de texto, pero con el texto en sí en blanco. Como con la barra de menú incrementamos el relleno alrededor del area de texto para dar un poco de espacio del borde del texto a otros elementos.

Example 11.8. Añadir contenidos del menú

```
/* create the menu */
{
    Ewl_Widget *file_menu = NULL, *item = NULL;

    /* create the file menu */
    file_menu = ewl_imenu_new(NULL, "file");
    ewl_container_child_append(EWL_CONTAINER(menu_bar), file_menu);
    ewl_widget_show(file_menu);

    /* add the open entry to the file menu */
    item = ewl_menu_item_new(NULL, "open");
    ewl_container_child_append(EWL_CONTAINER(file_menu), item);
    ewl_callback_append(item, EWL_CALLBACK_SELECT, file_menu_open_cb,
                                                        text_area);
    ewl_widget_show(item);

    /* add the quit entry to the file menu */
    item = ewl_menu_item_new(NULL, "quit");
    ewl_container_child_append(EWL_CONTAINER(file_menu), item);
    ewl_callback_append(item, EWL_CALLBACK_SELECT, destroy_cb, NULL);
    ewl_widget_show(item);
}
```

Ahora que el área de texto ha sido creada podemos proceder a crear las entradas del menú. He hecho esto dentro de su propio bloque para limitar el número de declaraciones al principio de la función, esto no es requerido por ninguna razón.

El menú es creado con una llamada a `ewl_imenu_new()`. Esta toma dos parámetros, el primero es la imagen a mostrar con este menú, en este caso NULL, siendo no imagen. El segundo parámetro es el nombre del menú tal como aparecerá en la barra del menú.

Una vez que el menú ha sido creado, podemos proceder a añadir entradas al menú por medio de una lla-

mada a `ewl_menu_item_new()`. Esta de nuevo toma dos parámetros, el icono a mostrar junto a esta entrada en el menú, y el nombre tal como aparecerá en el menú.

Conforme los elementos son añadidos al menú hacemos una llamada a `ewl_callback_append()` para conectar a la llamada EWL_CALLBACK_SELECT. La función dada será ejecutada cuando el usuario pulse en la entrada del menú. En el caso de "open" hemos pasado el área de texto al callback de open para permitirnos modificar fácilmente sus contenidos.

Otros menús podrían haber sido añadidos de la misma manera, pero para esta aplicación solo se requiere un menú.

Example 11.9. Vincular callbacks

```
    ewl_callback_append(main_win, EWL_CALLBACK_KEY_UP, key_up_cb, scroll);
}
```

Una vez todo está iniciado en la ventana principal añadimos los callbacks que deseamos recibir. En este caso estamos vinculándonos al callback EWL_CALLBACK_KEY_UP. No necesitamos hacer nada para que la rueda del ratón desplaze en el panel de desplazamiento puesto que está configurado en el propio panel de desplazamiento.

Example 11.10. callback de finalización

```
/* destroy the app */
static void destroy_cb(Ewl_Widget *win, void *ev, void *data) {
    ewl_widget_destroy(win);
    ewl_main_quit();
}
```

Cuando la ventana principal es cerrada destruímos el widget que es la ventana principal mediante una llamada a `ewl_widget_destroy()`.Cuando la ventana es destruida le decimos a Ewl que queremos salir llamando a `ewl_main_quit()`. Esto causará que Ewl pare el ciclo de proceso principal y la llamada previa a `ewl_main()` retornará.

Example 11.11. callback de open en el menú File

```
/* the file menu open button callback */
static void file_menu_open_cb(Ewl_Widget *win, void *ev, void *data) {
    Ewl_Widget *fd = NULL;
    Ewl_Widget *box = NULL;
    Ewl_Widget *home = NULL;

    /* create the file dialog window */
    fd_win = ewl_window_new();
    ewl_window_title_set(EWL_WINDOW(fd_win), PROG " -- file dialog");
    ewl_window_name_set(EWL_WINDOW(fd_win), PROG " -- file dialog");
    ewl_window_class_set(EWL_WINDOW(fd_win), PROG " -- file dialog");
    ewl_object_size_request(EWL_OBJECT(fd_win), 500, 400);
```

```
    ewl_object_fill_policy_set(EWL_OBJECT(fd_win),
                EWL_FLAG_FILL_FILL | EWL_FLAG_FILL_SHRINK);
    ewl_callback_append(fd_win, EWL_CALLBACK_DELETE_WINDOW,
                                destroy_filedialog_cb, NULL);
    ewl_widget_show(fd_win);

    /* fd win container */
    box = ewl_vbox_new();
    ewl_container_child_append(EWL_CONTAINER(fd_win), box);
    ewl_object_fill_policy_set(EWL_OBJECT(box),
                EWL_FLAG_FILL_FILL | EWL_FLAG_FILL_SHRINK);
    ewl_widget_show(box);

    /* the file dialog */
    fd = ewl_filedialog_new(EWL_FILEDIALOG_TYPE_OPEN);
    ewl_callback_append(fd, EWL_CALLBACK_VALUE_CHANGED, open_file_cb, data);
    ewl_container_child_append(EWL_CONTAINER(box), fd);

    /* add a home button */
    home = ewl_button_new("Home");
    ewl_callback_append(home, EWL_CALLBACK_CLICKED, home_cb, fd);
    ewl_object_fill_policy_set(EWL_OBJECT(home), EWL_FLAG_FILL_HFILL);
    ewl_container_child_append(EWL_CONTAINER(fd), home);
    ewl_widget_show(home);

    ewl_widget_show(fd);
}
```

Si un usuario pulsa en la entrada "open"del menú "file" la función `file_menu_open_cb()` será ejecutada. Cuando eso ocurra necesitamos crear el diálogo de archivo para que el usuario seleccione el archivo a ver.

De la misma manera que la ventana principal, creamos una ventana para contener el diálogo de archivo y le damos título, tamaño y clase. Requerimos un tamaño por defecto, seleccionamos su política de relleno y añadimos un callback para manejar la destrucción de la propia ventana. Entonces añadimos una caja simple en la ventana para contener el diálogo de archivo.

Una vez la ventana está configurada, hacemos la llamada para crear el diálogo de archivo. Esto es hecho con una llamada a `ewl_filedialog_new()`, especificando el tipo de diálogo de archivo que queremos crear. En este caso queremos un diálogo que nos permita abrir un fichero, así que especificamos EWL_FILEDIALOG_TYPE_OPEN. Podríamos haber especificado EWL_FILEDIALOG_TYPE_SAVE si quisiéramos usar el diálogo para grabar el archivo en lugar de abrirlo.

Entonces procedemos a crear un botón extra para permitir al usuario que navegue a su directorio personal con un simple click. Esto es hecho llamando a `ewl_button_new()` y empaquetando el botón subsiguiente en el propio diálogo de archivo.

Example 11.12. callback de finalización del diálogo de archivo

```
/* close the file dialog */
static void destroy_filedialog_cb(Ewl_Widget *win, void *ev, void *data) {
    ewl_widget_hide(win);
    ewl_widget_destroy(win);
}
```

Cuando necesitamos librarnos del diálogo de archivo eliminamos el widget de la pantalla con una llamada a `ewl_widget_hide()`, y una vez ya no es mostrado destruimos el widget con una llamada a `ewl_widget_destroy()`.

Example 11.13. callback del botón open del diálogo de archivo

```
/* the file dialog open button callback */
static void open_file_cb(Ewl_Widget *win, void *ev, void *data) {
    char *text = NULL;
    text = read_file((char *)ev);

    if (text) {
        ewl_text_text_set(EWL_TEXT(data), text);
        free(text);
    }
    text = NULL;

    ewl_widget_hide(fd_win);
}
```

Este callback será ejecutado cuando el usuario pulse el botón open en el diálogo de archivo, o cuando el usuario haga doble click en un archivo en un directorio. El evento pasado (el parámetro ev) será la ruta completa al archivo que el usuario haya seleccionado.

En nuestro caso, tomamos ese archivo y lo pasamos a la función para leer el archivo y devolver el texto del archivo. Entonces usando ese texto, siempre que esté definido, llamamos a `ewl_text_text_set()` que iniciará el texto del objeto de texto dado.

Dado que el usuario ha terminado ahora su selección el diálogo de archivo es escondido de la vista.

Example 11.14. callback del botón home del diálogo de archivo

```
/* the fd home button is clicked */
static void home_cb(Ewl_Widget *win, void *ev, void *data) {
    char *home = NULL;
    Ewl_Filedialog *fd = (Ewl_Filedialog *)data;

    home = getenv("HOME");
    if (home)
        ewl_filedialog_set_directory(fd, home);
}
```

Si el usuario hace click en el botón "Home" en el diálogo de archivo queremos mostrar los contenidos de su directorio personal. Ponemos el diálogo de archivo como datos al callback, así lo casteamos de nuevo a Ewl_Filedialog y capturamos el directorio personal del entorno. La llamada a `ewl_filedialog_set_directory()` cambia el directorio actual que está mostrando el diálogo de archivo para que sea el directorio personal del usuario.

Example 11.15. Leer el archivo de texto

```
/* read a file */
static char *read_file(char *file) {
    char *text = NULL;
    FILE *f = NULL;
    int read = 0, st_ret = 0;
    struct stat s;

    f = fopen(file, "r");
    st_ret = stat(file, &s);

    if (st_ret != 0) {
        if (st_ret == ENOENT)
            printf("not a file %s\n", file);
        return NULL;
    }

    text = (char *)malloc(s.st_size * sizeof(char));
    read = fread(text, sizeof(char), s.st_size, f);

    fclose(f);
    return text;
}
```

Esta es una rutina simple para tomar el archivo dado, abrirlo y leer sus contenidos en memoria. Probablemente no la mejor idea para una aplicación real, pero suficiente para este programa de ejemplo.

Example 11.16. callback de pulsación de teclado

```
/* a key was pressed */
static void key_up_cb(Ewl_Widget *win, void *ev, void *data) {
    Ewl_Event_Key_Down *e = (Ewl_Event_Key_Down *)ev;
    Ewl_ScrollPane *scroll = (Ewl_ScrollPane *)data;

    if (!strcmp(e->keyname, "q")) {
        destroy_cb(win, ev, data);

    } else if (!strcmp(e->keyname, "Left")) {
        double val = ewl_scrollpane_hscrollbar_value_get(EWL_SCROLLPANE(scroll));
        double step = ewl_scrollpane_hscrollbar_step_get(EWL_SCROLLPANE(scroll));

        if (val != 0)
            ewl_scrollpane_hscrollbar_value_set(EWL_SCROLLPANE(scroll),
                                                val - step);

    } else if (!strcmp(e->keyname, "Right")) {
        double val = ewl_scrollpane_hscrollbar_value_get(EWL_SCROLLPANE(scroll));
        double step = ewl_scrollpane_hscrollbar_step_get(EWL_SCROLLPANE(scroll));

        if (val != 1)
            ewl_scrollpane_vscrollbar_value_set(EWL_SCROLLPANE(scroll),
                                                val + step);

    } else if (!strcmp(e->keyname, "Up")) {
        double val = ewl_scrollpane_vscrollbar_value_get(EWL_SCROLLPANE(scroll));
        double step = ewl_scrollpane_vscrollbar_step_get(EWL_SCROLLPANE(scroll));

        if (val != 0)
```

```
            ewl_scrollpane_vscrollbar_value_set(EWL_SCROLLPANE(scroll),
                                                                val - step);

    } else if (!strcmp(e->keyname, "Down")) {
        double val = ewl_scrollpane_vscrollbar_value_get(EWL_SCROLLPANE(scroll));
        double step = ewl_scrollpane_vscrollbar_step_get(EWL_SCROLLPANE(scroll));

        if (val != 1)
            ewl_scrollpane_vscrollbar_value_set(EWL_SCROLLPANE(scroll),
                                                                val + step);
    }
}
```

`key_up_cb()` será llamada cuando el usuario suelte una tecla del teclado. El callback recibirá una estructura de evento Ewl_Event_Key_Down conteniendo la información de la propia pulsación de tecla. En nuestro caso solo necesitamos la entrada keyname que es el nombre de la tecla que fué pulsada.

Si el usuario presiona la tecla "q" simplemente llamamos al callback destroy y hemos acabado.

"Left", "Right", "Up" y "Down" se refieren a las teclas con flechas en el teclado. Si alguna de estas teclas es pulsada forzamos al panel de desplazamiento a que se desplace en la dirección especificada.

Para manipular el panel de desplazamiento necesitamos saber donde está este actualmente en el archivo y la cantidad de distancia que debería viajar cada incremento/decremento. Por suerte Ewl hace esto facil. La llamada a ewl_scrollpane_[hv]scrollbar_value_get() devolverá el valor actual de la barra de desplazamiento. Este es un valor double en el rango [0, 1] inclusivo. Un valor de 0 significa que la barra de desplazamiento está arriba y un valor de 1 que está abajo. Izquierda y derecha funcionan de la misma manera, con 0 siendo izquierda absoluta y 1 siendo derecha absoluta.

La segunda pieza de información es obtenida mediante la llamada a ewl_scrollpane_[hv]scrollbar_step_get(). El paso (step) es la distancia que viajará el panel de desplazamiento con una iteración. Así que usando esos dos valores podemos entonces mover la barra de desplazamiento en la dirección correcta con la llamada a ewl_scrollpane_[hv]scrollbar_value_set().

Example 11.17. Compilación

```
zero@oberon [ewl_intro] -< gcc -Wall -o ewl_text main.c \
  `ewl-config --cflags --libs`
```

Compilar una aplicación ewl es tan simple como llamar ewl-config y obtener los --cflags y --libs.

Eso es todo. Con eso deberías tener una aplicación Ewl funcionando plenamente incluyendo menús, un diálogo de archivo, y un área de texto con barras de desplazamiento horizontales y verticales. Este ejemplo tan solo araña la superficie de la potencia contenida dentro del conjunto de herramientas Ewl con muchos otros tipos de widget listos para usar.

Chapter 12. Evoak

evoak es un servidor de canvas. Esto es similar a un servidor X que sirve una pantalla y operaciones gráficas. Evoak sirve un único canvas para ser compartido por varias aplicaciones (clientes) permitiendo a cada cliente que manipule su propio conjunto de objetos en el canvas.

Receta: cliente hola Evoak

dan 'dj2' sinclair <zero@perplexity.org>

Esta receta es una introducción muy simple al mundo de la programación Evoak. Añadiendo a la gran tradición previa , muestra la verión canadiense de "Hello World" en un canvas evoak. canvas.

Example 12.1. Includes y pre-declaraciones

```
#include <Evoak.h>
#include <Ecore.h>

static unsigned int setup_called = 0;

static int canvas_info_cb(void *, int, void *);
static int disconnect_cb(void *, int, void *);
static void setup(Evoak *);
```

Obviamente necesitamos incluir la cabecera Evoak, y la cabecera Ecore es necesaria para tener acceso a las funciones callback.

Example 12.2. main

```
int main(int argc, char ** argv) {
    Evoak *ev = NULL;

    if (!evoak_init()) {
        fprintf(stderr, "evoak_init failed");
        return 1;
    }

    ecore_event_handler_add(EVOAK_EVENT_CANVAS_INFO, canvas_info_cb, NULL);
    ecore_event_handler_add(EVOAK_EVENT_DISCONNECT, disconnect_cb, NULL);

    ev = evoak_connect(NULL, "evoak_intro", "custom");

    if (ev) {
        ecore_main_loop_begin();
        evoak_disconnect(ev);
    }

    evoak_shutdown();
    return 0;
}
```

Evoak necesita ser iniciado al principio con una llamada a `evoak_init`. Esto iniciará todas las librerías internas y requerimientos para Evoak.

Siempre que Evoak se inicie correctamente, conectaremos dos callbacks, el primero es para información de canvas y el segundo por si somos desconectados del servidor Evoak. Estos serán discutidos mas tarde cuando los propios callbacks sean mostrados.

Una vez los callbacks están en su lugarnecesitamos conectar al servidor de canvas Evoak. Esto se hace mediante la llamada a `evoak_connect`. Los parámetros a `evoak_connect` son: el servidor al que conectar, el nombre del cliente, y la clase del cliente. Si el primer argumento es NULL, como en este ejemplo, el servidor Evoak por defecto es conectado. El segundo argumento a `evoak_connect` es el nombre de cliente, este valor debería ser único dado que será usado para distinguir el cliente de otros clientes. El argumento final, la clase, es el tipo del cliente, algunos de los valores posibles siendo: "background", "panel", "application" or "custom".

Si la llamada a `evoak_connect` falla un valor NULL será devuelto. Así, siempre que recibamos un objeto Evoak , empezamos el ciclo principal ecore. Cuando ecore termine llamamos `evoak_disconnect` para desconectarnos del servidor Evoak.

Terminamos llamando a `evoak_shutdown` para la limpieza final.

Example 12.3. callback de información de Canvas

```
static int canvas_info_cb(void *data, int type, void *ev) {
    Evoak_Event_Canvas_Info *e = (Evoak_Event_Canvas_Info *)ev;

    if (!setup_called) {
        setup_called = 1;
        setup(e->evoak);
    }
    return 1;
}
```

Se hará un callback de información de canvas cuando nuestro cliente reciba información del canvas del servidor Evoak. Con esta información podemos proceder a iniciar los contenidos de nuestros clientes. Esto está contenido en una opción setup?called dado que solo queremos inicializar una vez.

Example 12.4. callback de desconexión

```
static int disconnect_cb(void *data, int type, void *ev) {
    printf("disconnected\n");
    ecore_main_loop_quit();
    return 1;
}
```

El callback de desconexión será llamado cuando el cliente sea desconectado del servidor Evoak. En este caso, se usa la simple solución de salir.

Example 12.5. rutina de setup

```
static void setup(Evoak *ev) {
    Evoak_Object *o = NULL;

    evoak_freeze(ev);

    o = evoak_object_text_add(ev);
    evoak_object_text_font_set(o, "Vera", 12);
    evoak_object_color_set(o, 255, 0, 0, 255);
    evoak_object_text_text_set(o, "Hello Evoak, eh.");
    evoak_object_show(o);

    evoak_thaw(ev);
}
```

La rutina de setup será llamada una vez para iniciar la pantalla de nuestro cliente. Para este ejemplo, el cliente solo muestra el texto "Hello Evoak, eh".

La primera cosa que hacer es llamar `evoak_freeze`, esto debería protegernos de recibir callbacks indeseados mientras estamos iniciando nuestro interfaz. Al final de la función llamamos a su recíproca `evoak_thaw` para deshacer su efecto.

Entonces procedemos a crear un objeto de texto con `evoak_object_text_add` y tomando ese objeto de texto, iniciamos la fuente, el color, y los contenidos de texto con llamadas a `evoak_object_text_font_set`, `evoak_object_color_set`, y `evoak_object_text_text_set` respectivamente.

Example 12.6. Compilación

```
zero@oberon [evoak_intro] -> gcc -o hello_evoak main.c \
  `evoak-config --cflags --libs`
```

Como con tantas de las otras librerías basadas en EFL, compilar una aplicación Evoak es tan simple como llamar al programa evoak_config y obtener los contenidos de --cflags y --libs.

Eso es todo, esta fué una introducción realmente simple a Evoak y la superficie permanece sin ningún arañazo con respecto al potencial disponible para aplicaciones cliente.

Chapter 13. Emotion

Emotion es una librería de objetos de vídeo y media diseñada para interacuar con Evas y Ecore para proveer objetos "video" y "audio" que pueden ser movidos, redimensionados y posicionados como cualquier objeto normal, pero pueden reproducir video y audio y pueden ser controlados desde una API de control de alto nivel permitiendo al programador montar rápidamente un sistema multimedia con un trabajo mínimo. Emotion provee un sistema de capas de descodifición modular donde un módulo descodificador puede ser incorporado separadamente para proveer recursos de descodificación para Emotion. Emotion actualmente tiene un módulo descodificador que usa XINE como descodificador, permitiéndole reproducir DVDs, AVIs, MOVs, WMVs y muchos más. Su programa de prueba es ya un útil reproductor DVD (sin un bonito interfaz) y puede reproducir varias fuentes de vídeo con semi-translucencia y más.

Receta: Reproductor DVD rápido con Emotion

Carsten 'rasterman' Haitzler <raster@rasterman.com>

Para mostrar como hace de fácil Emotion poner un archivo de video, DVD, VCD u otro contenido de media en un canvas echa un vistazo al siguiente programa. Es un reproductor DVD completo, pero muy simple. Tiene unos controles de ratón limitados, no manejo de cambios de relación de aspecto, etc. Esto es todo de 55 líneas de código C.

Todo el código en esta receta y la próxima puede ser compilado usando:

Example 13.1. Compilando

```
$ gcc player.c -o player `emotion-config --cflags --libs`
```

Example 13.2. reproductor DVD en 55 lineas de código

```
#include <Evas.h>
#include <Ecore.h>
#include <Ecore_Evas.h>
#include <Emotion.h>

Evas_Object *video;

/* if the window manager requests a delete - quit cleanly */
static void
canvas_delete_request(Ecore_Evas *ee)
{
    ecore_main_loop_quit();
}

/* if the canvas is resized - resize the video too */
static void
canvas_resize(Ecore_Evas *ee)
{
    Evas_Coord w, h;
```

```
    evas_output_viewport_get(ecore_evas_get(ee), NULL, NULL, &w, &h);
    evas_object_move(video, 0, 0);
    evas_object_resize(video, w, h);
}

/* the main function of the program */
int main(int argc, char **argv)
{
    Ecore_Evas *ee;

    /* create a canvas, display it, set a title, callbacks to call on resize */
    /* or if the window manager asks it to be deleted */
    ecore_evas_init();
    ee = ecore_evas_software_x11_new(NULL, 0, 0, 0, 800, 600);
    ecore_evas_callback_delete_request_set(ee, canvas_delete_request);
    ecore_evas_callback_resize_set(ee, canvas_resize);
    ecore_evas_title_set(ee, "My DVD Player");
    ecore_evas_name_class_set(ee, "my_dvd_player", "My_DVD_Player");
    ecore_evas_show(ee);

    /* create a video object */
    video = emotion_object_add(ecore_evas_get(ee));
    emotion_object_file_set(video, "dvd:/");
    emotion_object_play_set(video, 1);
    evas_object_show(video);

    /* force an initial resize */
    canvas_resize(ee);

    /* run the main loop of the program - playing, drawing, handling events */
    ecore_main_loop_begin();

    /* if we exit the main loop we will shut down */
    ecore_evas_shutdown();
}
```

Ahora tenemos una introducción muy simple a Emotion. Este fragmento de código puede fácilmente ser expandido para trabajar con cualquier formato de media que emotion soporte, así como tratar relaciones de aspecto, navegación del teclado, y más.

Receta: Reproductor de media expandido con Emotion

Carsten 'rasterman' Haitzler <raster@rasterman.com>

Expandiendo nuestra receta previa, podemos hacer a emotion que maneje redimensionados apropiadamente (manteniendo la relación de aspecto)

Example 13.3. Reproductor de media Emotion

```
#include <Evas.h>
#include <Ecore.h>
#include <Ecore_Evas.h>
#include <Emotion.h>
```

```
Evas_Object *video;

/* if the window manager requests a delete - quit cleanly */
static void
canvas_delete_request(Ecore_Evas *ee)
{
    ecore_main_loop_quit();
}

/* if the canvas is resized - resize the video too */
static void
canvas_resize(Ecore_Evas *ee)
{
    Evas_Coord w, h;

    evas_output_viewport_get(ecore_evas_get(ee), NULL, NULL, &w, &h);
    evas_object_move(video, 0, 0);
    evas_object_resize(video, w, h);
}

/* the main function of the program */
int main(int argc, char **argv)
{
    Ecore_Evas *ee;

    /* create a canvas, display it, set a title, callbacks to call on resize */
    /* or if the window manager asks it to be deleted */
    ecore_evas_init();
    ee = ecore_evas_software_x11_new(NULL, 0, 0, 0, 800, 600);
    ecore_evas_callback_delete_request_set(ee, canvas_delete_request);
    ecore_evas_callback_resize_set(ee, canvas_resize);
    ecore_evas_title_set(ee, "My DVD Player");
    ecore_evas_name_class_set(ee, "my_dvd_player", "My_DVD_Player");
    ecore_evas_show(ee);

    /* create a video object */
    video = emotion_object_add(ecore_evas_get(ee));
    emotion_object_file_set(video, "dvd:/");
    emotion_object_play_set(video, 1);
    evas_object_show(video);

    /* force an initial resize */
    canvas_resize(ee);

    /* run the main loop of the program - playing, drawing, handling events */
    ecore_main_loop_begin();

    /* if we exit the main loop we will shut down */
    ecore_evas_shutdown();
}
```