# Livro de Receitas EFL

**por Varios e Ben 'technikolor' Rockwood**

# Livro de Receitas EFL

por Varios e Ben 'technikolor' Rockwood

# Índice

# Lista de Exemplos

# Capítulo 1. Introdução

Bem-vindo ao estado iluminado da programação. Este livro é uma coleção de idéias, dicas, tutoriais e introduções arranjadas no estilo receita com o objetivo de ajudar você a se tornar rapidamente hábil com as Enlightenment Foundation Libraries (Bibliotecas de Fundação do Enlightenment) . As Enlightenment Foundation Libraries, chamadas simplesmente de EFL, são uma coleção de bibliotecas originalmente escritas para suportar o gerenciador de janelas Enlightenment DR17. No entanto, ao passo que cresciam e iam sendo testadas, foram adicionadas funcionalidades gerais que levou-nos a desfrutar de um conjunto de bibliotecas ricas e poderosas que podem resolver todo tipo de problemas e também atuar como uma venerável alternativa para as atualmente populares GTK e QT.

As EFL são um grupo amplo de bibliotecas C que podem resolver uma grande quantidade de necessidade gráficas em diversas plataformas. A seguir uma análise concisa das bibliotecas que fazem parte da EFL.

### Lista de componentes EFL

| | |
|---|---|
| Imlib2 | Biblioteca completa de manipulação de imagem, incluindo renderização de drawables X11. |
| EVAS | Biblioteca de canvas com vários motores de backend incluindo aceleração de hardware via OpenGL. |
| Ecore | Coleção modular de bibliotecas com controle de eventos e temporizadores. Inclui também sockets, IPC, setup de FB e X11, controle de eventos, controle de job, controle de configuração, e mais. |
| EDB | Uma biblioteca de base de dados capaz de armazenar strings, valores e dados para controle de configuração simples, consistente e centralizado. |
| EET | Un formato flexivel de container para armazenar imagens binárias e dados. |
| Edje | Uma biblioteca e toolkit de abstração de imagens baseado no EVAS, mantendo todos os aspectos de interface de usuário completamente separados do código da aplicação usando mensagens. |
| Embryo | Uma linguagem de script típicamente usanda com Edje para controle avançado. |
| Etox | Uma biblioteca de formatação e manipulação de texto, completa com capacidades de estilização de texto. |
| Esmart | Uma biblioteca consistente de vários objetos inteligentes EVAS para fácil reutilização, incluindo o popular hack de transparência. |
| Epeg | Uma biblioteca de thumbnailing rápida como um raio para imagens JPEGs independente de outros componentes EFL. |
| Epsilon | Uma biblioteca de thumbnailing flexivel e rápida para PNG, XCF, GIF and JPEG. |
| Evoak | Uma biblioteca e toolkit de servidor canvas EVAS. |
| EWL | Uma completa biblioteca de widgets. |
| Emotion | Uma biblioteca de objetos EVAS para reprodução de vídeo e DVD utilizando libxine. |

# Capítulo 2. Imlib2

Imlib2 é a sucessora da Imlib. Não é apenas uma nova versão - é uma biblioteca completamente nova. Imlib2 pode ser instalada junto com a Imlib 1.x já que são efetivamente bibliotecas diferentes - mas elas tem funcionalidade similar.

Imlib2 pode fazer o seguinte:

- Carregar imagens do disco em um dos muitos formatos
- Salvar imagens no disco em um dos muitos formatos
- Renderizar dados de imagem dentro de outras imagens
- Renderizar imagens em uma "drawable X"
- Produzir pixmaps e mascaras de pixmap de imagem
- Aplicar filtros em imagens
- Rotacionar imagens
- Aceitar dados RGBA para imagens
- Scalar imagens
- Alpha blend de imagens em outras imagens ou "drawables"
- Aplicar correção de cor, tabelas de modificação e fatores na imagem
- Renderizar imagens sobre imagens com correção de cor e tabelas de modificação
- Renderizar texto truetype com anti-aliasing
- Renderizar texto truetype con anti-aliasing em qualquer ângulo
- Renderizar linha com anti-aliasing
- Renderizar retangulo
- Renderizar gradientes linerares mult-cores
- Dispor inteligentemente os dados na memória para o máximo de rendimento
- Alocar cores automaticamente
- Permitir controle completo sobre o cache e alocação de cor
- Fornecer instruções assembly MMX altamente otimizado para rotinhas centrais
- Fornecer interface para plug-in de filtros
- Fornecer interface de carga e gravação de imagens por plug-ins em tempo de execução
- A biblioteca de composição, renderização e manipulação mais rápida para o X

Se o que você quer fazer não está na lista acima, então provavelmente a Imlib2 não faz. Se faz, provavelmente irá fazer mais rápido que qualquer outra biblioteca que encontrar (isto inclui dk-pixbuf, gdkrgb, etc.) principalmente por causa do código altamente otimizado e um subsistema inteligente que faz o trabalho sujo pra você e separa as partes que causam tédio deixando que a Imlib2 faça todas as otimizaçõões pra você.

Imlib2 fornece um potente motor para manipulação e renderização de imagem. Usando carregadores, o Imlib2 pode manejar uma variedade de formatos incluindo BMP, GIF (via unGIF), JPEG, PNG, PNM, TGA, TIFF, XPM entre outros.

# Receita: marca d'agua para imagens

Ben technikolor Rockwood <benr@cuddletech.com>

Com tanta gente publicando tantas imagens online é fácil esquecer de onde elas vieram e difícil assegurar que material com direitos de cópia não seja inadivertidamente mal utilizado. Simplesmente adicionando uma marca d'agua, como um logo do seu site, em cada uma das imagems pode resolver ambos problemas. Mas adicionar marcas d'agua manualmente é uma tarefa longa e repetitiva. Imlib2 pode fácilmente ser usada para resolver este problema. O que precisamos fazer é pegar uma imagem de entrada, e então espeficicar uma imagem para a marca d'agua (seu logo), posicionar a marca d'agua na imagem de entrada e então salvar em uma nova imagem que usaremos no site. A aplicação seria algo parecido com isto:

### Exemplo 2.1. Programa Marca d'agua com Imlib2

```
#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);
}
```

Vendo o exemplo, primeiro fazemos uma checagem básica dos argumentos, aceitando uma imagem de entrada como primeiro argumento e um nome de imagem de saída para nossa cópia com a marca d'agua. Usando `imlib_load_image()` carregamos a imagem de entrada e então obtemos as suas dimensões usando as funções get. Com a função `imlib_clone_image()` podemos criar uma cópia da imagem de entrada, que será a base da nossa imagem com a marca d'agua de saída. Depois carregamos la imagem que será a marca d'agua, e observe que usamos

`imlib_context_set_image()` para mudar o contexto de imagem de entrada (image_input) para a imagem de marca d'agua (image_watermark). Agora obemos as dimensões da imagem também. No bloco final fazemos um simples cáculo para determinar a posicionamento da marca d'agua na imagem de saída, neste caso quero que a marca d'agua fique no canto direito inferior. A função mágica que faz o trabalho real neste programa é `imlib_blend_image_onto_image()`. Observe que mudamos o contexto da imagem de saída antes de continuar. A função blend irá, como seu próprio nome sugere (to blend = misturar), mistura duas imagens juntas que referimos como imagem fonte e imagem de destino. A função blend mistura uma imagem fonte sobre a imagem no contexto atual que nós designamos como o destino. Os argumentos passados para `imlib_blend_image_onto_image()` podem parecer estranhos, precisamos dizer que fonte usar (a marca d'agua), se quer mesclar com canal alfa (0 para não), as dimensões da imagem fonte (x, y, w, h), as dimensões da imagem de destino (x, y, w, h). Você notará que no exemplo colocamos as posições x, y da imagem fonte (marca d'agua) como 0 e usamos a largura total. O destino (imagem de entrada) se coloca no canto inferior direito menos as dimensões da marca d'agua, então especificamos a largura e altura da marca d'agua. Finalmente, usamos a função `imlib_save_image()` para salvar a imagem de saída.

Enquanto este exemplo deve ser melhorado significativamente para uso real, ele demostra a base de mesclagem da Imlib2 para resolver eficientemente um problema muito comum.

# Receita: Escalonando imagem

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

Conforme mais pessoas obtem a habilidade de publicar imagens na internet é muitas vezes desejado escalonar estas imagens para um tamanho menor para reduzir o uso de banda. Isto pode ser facilmente resolvido usando um simples programa Imlib2.

Esta receita pega o nome da imagem de entrada, a nova largura, altura e o nome da imagem de saída, então escala a imagem de entrada para os valores dados, salvando-a na imagem de saída.

### Exemplo 2.2. Escalando imagem

```
#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;
}
```

No exemplo ha uma mínima checagem dos argumentos, simplesmente checa se o número de argumentos passado está correto.

A imagem fonte é carregada com uma chamada à `imlib_load_image()`, do qual carregará os dados da imagem na memória. Se a chamada falhar, NULL será retornado. Uma vez com os dados da imagem precisamos selecionar a imagem para ser o contexto atual. Isto diz ao Imlib2 em qual imagem se efetuará as operações. Isto é feito chamando `imlib_context_set_image()`. Uma vez que a imagem foi definida como contexto atual podemos continuar com o escalonamento. Isto é feito chamando `imlib_create_cropped_scaled_image()`, que pega como argumentos, a posição de inicio em x, y, largura e altura da imagem fonte, e a largura e altura da imagem escalada. A razão de passarmos a informações da imagem fonte é que esta função pode recortar a imagem se assim desejar. Para recortar, simplesmente modifica-se o x, y, largura da fonte e altura da fonte desejado. Isto resultará em uma nova imagem out_img sendo produzida. Se o escalonamento falhar, será devolvido NULL . Nós então definimos a out_img para ser o contexto atual e chamamos a função de gravar, `imlib_save_image()`.

Apesar de simples, este programa mostra a simplicidade de escalonamento de imagem usando a API da Imlib2.

# Receita: Rotação Livre

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

Algumas vezes é desejado rotacioanr uma imagem em um ângulo específico. Imblib2 faz este processo fácilmente. Este exemplo mostra como fazer isto. Se você deseja rotacionar a imagem em angulos de 90 graus, veja a receita de roação de 90 graus já esta receita deixa uma borda negra ao redor da imagem.

### Exemplo 2.3. Rotação livre

```
#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;
}
```

Antes uma checagem simples dos argumentos. Começamos carregando a imagem específicada na memoria com `imlib_load_image()` passando o nome da imagem como parâmetro. Então pegamos a imagem fazendo-a como conexto atual com `imblib_context_set_image`. Os contextos sãos usados na Imlib2 para saber qual imagem trabalhar. Quando você quer fazer chamada imlib em uma imagem, a imagem deve estar selecionada como contexto atual. Então convertemos o ângulo dado em graus para radiano já que a função de rotação da Imlib2 trabalha em redianos. A rotação é feita com `imlib_create_rotated_image()`. A função de rotação irá retornar a nova imagem. Para salvar a nova imagem precisamos seleciona-la como contexto atual, e novamente chamar `imlib_context_set_image()`. Então uma simples chamada à `imlib_save_image()` dando o nome do arquivo de saída salva a nova imagem rotacionada.

A função de rotação da Imlib2 colocará uma borda preta em volta da imagem para preencher qualquer espaço vazio. Esta borda é calculada de maneira que a imagem rotacioanda caiba na saída. Isto colocará bordas ao redor da imagem de saida se você rotacionar em 180 graus.

# Receita: Rotação de imagen em 90 graus

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

Com uma câmera digital as vezes é desejado rotacionar sua imagem em 90, 180 ou 270 graus. Esta receita mostrará como fazer isto fácilmente com a Imlib2. Esta receita também não colocará bordas pretas ao redor da imagem como visto no exemplo de ratação livre.

### Exemplo 2.4. rotação de imagem em 90 graus

```
#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;
}
```

Antes uma simples checagem de erro, nós carregamos a imagem a ser rotacionada com uma chamada à `im-`

`lib_load_image()`. Esta função aceita um nome de arquivo e retorna um objeto Imlib_Image, ou NULL no caso de erro ao tentar carregar. Uma vez carregada a imagem é definida como o contexto atual, a imagem que a Imlib2 fará todas as operações, com `imlib_context_set_image()`. A rotação é feita atravéz da chamada à `imlib_image_orientate()`. O parâmetro para _orientate muda a quantidade de rotação. Os valores possíveis são: [1, 2, 3] que siginifica a rotação no sentido horário de [90, 180, 270] graus respectivamente. Uma vez rotacionada chamamos `imlib_save_image()` informando o nome do arquivo para a nova imagem para que o Imlib2 grave a imagem rotacionada.

Com este exemplo em mãos você deve estar habilitado à rotacionar imagens rápidamente em intervalos de 90 graus usando Imlib2.

# Receita: Reflexão de Imagem

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

A Imlib2 contém funções para fazer reflexão de imagem. Isto pode ser feito horizontal, vertical ou diagonalmente. Esta receita mostrará como implementar esta funcionalidade.

### Exemplo 2.5. Reflexão de Imagem

```
#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 exemplo faz uma mínima checagem de argumentos, então carrega a imagem usando `imlib_load_image()` passando o nome do arquivo. `imlib_load_image()` devolveré um objeto Imlib_Image, ou NULL se falhar. Uma vez com o objeto de imagem selecionamos-o como o contexto atual com a chamada à `imlib_context_set_image()`. Isto diz ao Imlib2 que esta é a imagem que nós queremos trabalhar e todas as operações serão feitas nela. Com o contexto de imagem configurado decidimos o tipo de reflexão que queremos efetuar. Isto é feito com uma das chamadas: `imlib_image_flip_horizontal()`, `imlib_image_flip_vertical()`, e `imlib_image_flip_diagonal()`. A reflexão diagonal essencialmente pega o canto superior esquerdo e a faz como canto inferior direito. O canto superior direito se transforma no canto inferior esquerdo. Uma vez feita a reflexão na imagem chamamos `imlib_save_image()` para salvar informando o nome do novo arquivo, e assim terminamos.

Isto deve fornecer um exemplo de imagem reflexionada com Imlib2. Este exemplo necessitará de aperfeiçoamentos antes de ser posto como uma aplicação real mas o básico está aí.

# Capítulo 3. EVAS

Evas é uma API de canvas com aceleração por hardware que pode desenhar texto com anti-aliasing, imagens suaves super e sub-sampleadas, alpha-blend, assim como limitar-se à utilizar primitivas X11 normais como pixmap, linhas e retângulos com velocidade se sua CPU ou hardware gráfico são muito lentos.

Evas abstrai qualquer necessidade de conhecimento avançado sobre as características do display do seu XServer, qual a tonalidade de cor ou quais visuais mágicas ele tem. O maximo que você precisa informar ao Evas é quantas cores ( no máximo ) utilizar se o seu display não for truecolor. Por padrão sugere-se usar 216 cores ( isto equivale a um cubo 6x6x6 - exatamente é o mesmo cubo de cor utilizado pelo Netscape, Mozilla, gdkrgb, etc. de maneira que as cores serão compartilhadas). Se Evas não pode alocar suficiente cores, ele continuará reduzindo o tamanho do cubo de cor até alcançar branco e preto. Desta maneira, o Evas pode mostrar qualquer coisa em um terminal preto e branco, VGA 16 cores, 256 cores e 15, 16, 24 e 32bit de cores.

# Receita: Usando Ecore_Evas para simplificar a inicialização de canvas X11

Ben technikolor Rockwood

Evas é uma biblioteca potente e simples de usar, mas antes de estabelecer um canvas, um drawable X11 deve ser configurado. Configurar manualmente o X11 pode ser uma tarefa caótica e frustrante que te impossibilita se concentrar no que realmente deseja fazer: desenvovler uma aplicação Evas. Mas tudo isto pode ser evitado usando o módulo Ecore_Evas do Ecore para fazer todo o trabalho pesado por você.

O seguinte exemplo é um template básico que pode ser usando como ponto de partida para qualquer aplicação Evas que desejar construir, cortando significativamente o tempo de desenvolvimento.

### Exemplo 3.1. Template 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;
}
```

Um detalhe completo sobre Ecore_Evas pode ser econtrado no capítulo de Ecore neste livro, mas este básico template deve permitir você brincar com Evas imediatamente. As chamdas importantes para observar são `ecore_evas_borderless_set()` que define se a janela Evas está controlada pelo seu gerênciador de janelas ou sem bordas, e `evas_font_path_append()` que define o(s) caminho(s) das fontes utilizadas por sua aplicação Evas.

# Receita: Vínculos de teclado, usando eventos de teclado Evas

Ben technikolor Rockwood

Muitas aplicações pode se beneficiar por fornecer vínculos de teclado para operações comumente usadas. Quer aceitando texto de forma que a EFL não espera, ou apenas um jeito de vincular a tecla + para aumentar o volume de um mixador, os vinculos de teclado (keybinds) pode adicionar pequenas funcionalidades fazendo sua aplicação um sucesso.

O seguinte código é uma aplicação simples e completa, útil para explorar keybinds usando callbacks de evento EVAS. O exemplo criará numa janela de cor preta de 100 por 100 pixels do qual pode-se pressionar teclas.

### Exemplo 3.2. Captura do 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;
}
```

Você pode compilar este exemplo da seguinte maneira:

### Exemplo 3.3. Compilação de keybinds EVAS

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

Neste exemplo o canvas é iniciado de maneira habitual usando Ecore_Evas para fazer o trabalho sujo. A mágica ocorre no callback `evas_object_event_callback_add()`.

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

Adicionando um callback à base_rect, que atua como um backgound do canvas, nós podemos executar uma função ( neste caso a `key_down()` ) quando encontramos um evento de tecla pressionada, EVAS_CALLBACK_KEY_DOWN, conforme definido em `Evas.h`.

Há uma coisa muito importante para se fazer em adição a definição do callback: capturar o foco. A função `evas_object_focus_set()` captura o foco num objeto Evas dado. Que é o objeto que tem o foco que aceitará os eventos sempre quando você explicitamente define o objeto Evas do qual o callback está vinculado. E só um objeto pode ter o foco de cada vez. O maior problema comumente encontrado com os callbacks Evas é esquecer de capturar o 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);
}
```

A função `key_down()` é chamda toda vez que um evento de tecla pressionada ocorre depois de definir seu callback. A declaração da função é a mesma de um callback Evas comum ( ver `Evas.h` ). A parte importante da informação que nós precisamos saber é qual tecla foi pressionada, que está contida na estrutura `event_info` Evas. Antes de inciar a estrutura `Evas_Event_Key_Down` para usar como visto acima podemos acessar o elemento `keyname` para determinar qual tecla foi pressionada.

Na maioria dos casos você provavelmente usará um `switch` ou `if`s alinhados para definir quais teclas fazem o que, e recomenda-se que esta funcionalidade seja complementada com uma configuração EDB para fornecer centralização e fácil expansão da configuração de keybinds das suas aplicações.

# Receita: Introdução aos objetos inteligentes Evas

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

Quando trabalhar mais com Evas, você terá vários `Evas_Object` dos quais estará trabalhando e aplicando as mesmas operações para mantê-los em sincronia. Seria mais conveniente agrupar todos este `Evas_Object` em um único objeto onde todas as transformações podem ser aplicadas.

Os objetos inteligentes Evas fornecem a capacidade de escrever seus próprios objetos e ter a Evas chamando suas funções para movimentar, redimencionar, esconder, fazer camadas e todas as coisas que um `Evas_Object` é responsável de controlar. Junto com os callbacks normais do `Evas_Object`, os objetos inteligentes permitem definir suas próprias funções para controlar quaisquer operações especiais que desejar.

Esta introdução está dividia em 3 arquivos: foo.h, foo.c e main.c. O objeto inteligente criado chama-se `foo` e está definido nos foo.c e foo.h, main.c mostra como o novo objeto inteligente pode ser usado.

O objeto inteligente em si é apenas dois quadrados, um dentro do outro, onde o interno está separado 10% da borda do quadrado externo. Ao passo que o programa principal se executa um callback de temporizador Ecore reposicionará e redimencionará o objeto inteligente.

O arquivo básico para este objeto inteligente vem de um template Evas Smart Object de autoria de Atmos e localizada em: `www.atmos.org/code/src/evas_smart_template_atmos.c` `[http://www.atmos.org/code/src/evas_smart_template_atmos.c]` que foi baseado no template de autoria de Rephorm.

Primeiro precisamos definir a interface externa do nosso objeto inteligente. Neste caso apenas precisamos de uma chamada para criar o novo objeto.

### Exemplo 3.4. foo.h

```
#ifndef _FOO_H_
#define _FOO_H_

#include <Evas.h>

Evas_Object *foo_new(Evas *e);

#endif
```

Com isto fora do caminho, entramos na escura entranhas da besta, o código do objeto inteligente.

### Exemplo 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` armazenará toda a informação do nosso objeto. Neste caso é o objeto caixa externa, caixa interna, um objeto de clipping e a atual posição e tamanho do 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);
```

As pré-declarações requeridas para o objeto inteligente. Estas serão explicadas quando chegarmos na implementação.

```
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()` única interface externa e é responsável por configurar o objeto em si. A chamada à `foo_object_new()` fará o trabalho pesado na criação do objeto. A `evas_object_smart_data_get()` é mais uma checagem de erro do que qualquer outra coisa. Quando executa-se `foo_object_new()` adicionará o objeto inteligente ao evas e isto resultará em uma chamada à `add`. Neste caso `add` criará um `Foo_Object`. Assim, estamos apenas certificando que `Foo_Object` foi criado.

```
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;
}
```

Nossa função `foo_object_new()` tem uma simples tarefa de adicionar nosso objeto inteligente num Evas dado. Se faz isto através de `evas_object_smart_add()` passando o Evas e o objeto `Evas_Smart *`. Nosso `Evas_Smart *` é produzido por chamar `_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;
}
```

Você irá observar que `Evas_Smart *smart` nesta função é declarado como `static`. Porque não importa quantos `Evas_Object` criemos, só haverá um único objeto `Evas_Smart`. Como Raster enfatiza, `Evas_Smart` é como uma definição de classe C++, não uma instância. O `Evas_Object` é a instância do `Evas_Smart`.

O objeto inteligente em si é criado atravéz de uma chamada à função `evas_smart_new()`. Para esta função passamos o nome do objeto inteligente, todas as rotina de callback para o objeto inteligente, e qualquer dado de usuário. Neste casso nós não temos dados de usuário, então 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);
}
```

Quando `evas_object_smart_add()` é chamada em `foo_object_new()`, esta função, `_foo_object_add()`, será chamada de modo que podemos inicializar qualquer dado interno para este objeto inteligente.

Para este objeto inteligente nós inicializamos três `Evas_Object` internos. Sendo estes `data->clip`usado para fazer

clipping com os outros dois objetos, `data->outer` que é nosso retângulo externo e `data->inner`, nosso retângulo interno. Os retângulos interno e externo tem o clipping setado para o objeto clip e são mostrados imediatamente. O objeto clip não é mostrado, só será mostrando quando o usuário chamar `evas_object_show()` neste objeto.

Finalmente chamamos `evas_object_smart_data_set()` para setar nosso novo `Foo_Object` como dado para este objeto inteligente. Este dado será recuperado em outras funções deste objeto chamando 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);
    }
}
```

O callback `_foo_object_del()` será executado se o usuário chamar `evas_object_del()` em nosso objeto. Para este objetos é simples, basta chamar `evas_object_del` no nossos 3 retângulos e liberar a nossa estrutura `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 funções: `_foo_object_layer_set()`, `_foo_object_raise()`, `_foo_object_lower()`, `_foo_object_stack_above()`, e `_foo_object_stack_below()` trabalham todos da mesma maneira, aplican-

do a função requerida `evas_object_*` no objeto `data->clip`.

Estas funções são disparadas pelo uso de: `evas_object_layer_set()`, `evas_object_raise()`, `evas_object_lower()`, `evas_object_stack_above()`, e `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;
    }
}
```

O callback `_foo_object_move()` será disparado quando `evas_object_move()` é chamada em nosso objeto. Cada um dos objetos interno é movido na posição correta com chamadas para `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;
    }
}
```

O callback `_foo_object_resize()` será disparado quando o usuário chama `_foo_object_resize()` no nosso objeto. Assim, para nosso objeto, precisamos redimensionar `data->clip` e `data->outer` ao tamanho completo disponível para nosso objeto. Isto é feito com as chamadas à `evas_object_resize()`. Então precisamos mover e redimencionar o objeto `data->inner` para que permaneça na posição correta no retângulo externo. Isto é feito com `evas_object_move()` e `evas_object_resize()` respectivamente. Então guardamos a largura e altura atual do nosso objeto para que possamos referência-los mais 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);
}
```

O callback `_foo_object_show()` será disparando quando `evas_object_show()` for chamada no nosso objeto. Para mostrar nosso objeto todo o que precisamos fazer é mostrar a região de clipe, já que o retângulo atual está "cli-

pado". Isto é feito chamando `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);
}
```

O callback `_foo_object_hide()` será disparado quando uma chamada à `evas_object_hide()` for feita no nosso objeto. Por estarmos usando um objeto de clipping interno só precisamos esconder o objeto clip, `data->clip`, para esconder nosso objeto inteligente. Isto é feito por meio da chamada à `evas_object_hide()`.

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

A função `_foo_object_color_set()` será disparada quando `evas_object_color_set()` for chamada por nosso `Evas_Object`. Mas, já que não quero mudar as cores do meu objeto, 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);
}
```

O callback `_foo_object_clip_set()` será disparada quando `evas_object_clip_set()` for chadamda por nosso objeto. Neste caso propagamos isto a nosso objeto de clipping interno por meio da chamada à `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);
}
```

O callback `_foo_object_clip_unset()` será disparado quando uma chamada a `evas_object_clip_unset()` for feita no nosso objeto. Nós simplesmente removemos o clip interno por meio da chaamda á `evas_object_clip_unset()`.

Uma vez completo nossoo código do objeto inteligente, podemos criar o programa principal para utilizar o novo objeto inteligente.

### Exemplo 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;
}
```

Muito deste programa é parte da receita usando Ecore_Evas dado anteriormente. A criação de nosso novo objeto inteligente e vista no 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);
```

Uma vez que a sua nova `foo_new()` devolve nosso objeto, podemos manipula-lo com as chamadas normais Evas, e assim ajustarmos a posição, tamanho, camada e então mostrar o objeto.

Uma vez criado e mostrado o novo objeto inteligente, iniciamos um temporizador Ecore para disparar a cada 0.1 se-

gundos. Quando o temporizador disparar ele irá executar a função `timer_cb()`. Esta função callback diminuirá e aumentará o tamanho de nosso objeto inteligente enquanto o mantém centralizado na janela principal.

Compilar este exemplo é tão simples como:

### Exemplo 3.7. Compilação

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

Os objetos inteligentes Evas são simples de criar mas fornecem um potente mecanismo para abstrair partes do seu programa. Para ver mais objetos inteligente dê uma olhada em qualquer dos objetos Esmart, Etox ou Emotion.

# Capítulo 4. Ecore

O que é Ecore? Ecore é o núcleo central da camada de abstração de eventos e abstração X que permite fazer seleções, Xdnd, coisas do X em geral, loops de evento, controle de expiração e tempo de inatividade de uma maneira rápida, otimizada e conveniente. É uma biblioteca separada de maneira que qualquer um pode usar suas tarefas nas aplicações de um modo fácil.

Ecore é totalmente modular. Em sua base está os controladores de eventos, temporizadores, e funções de inicialização e encerramento. Os módulos de abstração Ecore incluem:

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

Ecore é tão modular e potente que pode ser muito útil no desenvolvimento de programas não gráficos. Por exemplo, vários servidores web estão sendo escritos só com Ecore e o módulo Ecore_Con para abstrair a comunicação por socket.

# Receita: Introdução ao Ecore Config

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

O módulo Ecore_Config provê ao programador uma jeito muito simples de preparar arquivos de configuração para seu programa. Esta receita dará um exemplo de como integrar os principios do Ecore_Config em seu programa e utilizar-lo para conseguir os dados de configuração.

### Exemplo 4.1. Um simples programa 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 pode ver, o uso básico do Ecore_Config é simples. O sistema é inicializado com a função `ecore_config_init`. O nome do programa indica onde o Ecore_Config irá procurar pela sua base de dados de configuração. O diretório e o nome do arquivo são: ~/.e/apps/NOME_DO_PROGRAMA/config.db.

Para cada variável de configuração que você receber do Ecore_Config, pode assinar um valor padrão no caso do usuário não ter um arquivo config.db. Os valores por padrão são assinados com ecore_config_*_default onde * é um dos tipos Ecore_Config. O primeiro parâmetro é a chave que vai ser acessada. Estas chaves devem ser únicas em seu programa. O valor passado é do tipo apropriado para esta chamada.

A chamada a `ecore_config_load` irá ler os valores do arquivo config.db no Ecore_Config. Depois disto, podemos acessar os arquivos com os métodos ecore_config_*_get (novamente * é o tipo de dados desejado). Estas rotinas usam do nome da chave para este item e retornam o valor associado à chave. Cada função retorna um tipo que corresponde ao nome da chamada da função.

`ecore_config_shutdown` é então chamada para terminar o sistema Ecore_Config antes de sair do programa.

### Exemplo 4.2. Comando para comilação

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

Para compilar o programa você pode usar o script ecore-config para obter toda informação de biblioteca e linkagem requerida pelo Ecore_Config. Se você executar este programa como está, receberá como saída os valores passados pelo ecore_config. O programa uma vez funcionando, você pode criar um simples arquivo config.db para ler os valores.

### Exemplo 4.3. Um simples script 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
```

Quando for executado, o build_cfg_db.sh criará um arquivo config.db no diretório atual. Este arquivo pode ser compiado para ~/.e/apps/NOME_DO_PROGRAMA/config.db onde NOME_DO_PROGRAMA é o valor passado para ecore_config_init. Uma vez copiado para o diretório, executando o programa test mostrará os valores do arquivo config no lugar dos valores padrão. Você pode especificar quantas chaves de configuração desejar no arquivo config e o Ecore_Config mostrará o valor do usuário ou o valor padrão.

# Receita: Listeners Ecore Config

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

Quando se usa Ecore Config para controlar a configuração da sua aplicação é bom ser notificado quando esta configuração for modificada. Consegue-se isto mediante o uso de listeners Ecore_Config.

### Exemplo 4.4. Ecore_Config listener

```
#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 é inicializado da maneira habitual, e criamos algumas chaves padrão como normalmente ocorre. As partes interessantes entram no jogo com as chamadas para `ecore_config_listen()`. Esta é a chamada que diz para o Ecore_Config que desejamos ser notificados das mudanças da configuração. `ecore_config_listen()` pega 5 parâmetros:

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

O campo `name` é uma string dado por você para identificar este callback de listener. O campo `key` é o nome da chave que deseja escutar, este será o mesmo nome da chave passado para as chamadas _default acima. O `listener callback` é a função callback que será executada em caso de mudança. O `id tag` é uma etiqueta numérica que pode ser dada a cada listener e será passada para a função callback. Finalmente, `user data` é qualquer dado que deseja ser passado para a callback quando este é executado.

A função callback tem uma assinatura semelhante a:

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

O campo `key` é o nome da chave que quer escutar. O parâmetro `type` conterá o tipo Ecore_Config. Este pode ser um dos:

| | |
|---|---|
| PT_NIL | Propriedade sem valor |
| PT_INT | Prpriedade inteira |
| PT_FLT | Propriedade Float |
| PT_STR | Propriedade String |
| PT_RGB | Propriedade Colour |
| PT_THM | Propriedade Theme |

O parâmetro `tag` é o valor que foi passado acima, na chamada de criação de listener. Finalmente, `data` é qualquer dado anexado ao listener quando este foi criado.

Se desejar remover o listener em um momento posterior use `ecore_config_deaf()`. Que tem três parâmetros:

- name

- key
- listener callback

Cada um dos parâmetros corresponde ao parâmetro dado na chamada inicial `ecore_config_listen()`.

### Exemplo 4.5. Compilação

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

Se você executar o programa verá os valores padrões na tela. Se você executar agora examine como segue:

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

(foo é o nome passado para `ecore_config_init()`). Você deve então ser capaz de modificar a configuração da sua aplicação e, após digitar 'save', ver no console os valores modificados.

# Receite: Conexão para um servidor com Ecore_Con

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

Client/server estão tornando-se completamente comuns hoje em dia. Para este fim, Ecore pode simplifcar sua vida. Ecore tem o subsistema Ecore_Con que controla toda chatisse de conectar com servidores e de clientes conectados no seu servidor.

Esta receita mostra um exemplo conectando com um outro servidor e recebendo alguma informação de volta. Nós iremos conectar com o website do enlightement e pegar uma das páginas.

Agora, antes de você ficar confuso. A terminologia Ecore_Con pode semear um pouco de confusão embaralhando seu cérebro. Quando você está conectando com um servidor estará trabalhando com as chamadas de servidor do ecore_con. Isto inclue os callbacks de servidor.

Lembrando que estou fazendo poucas checagem dos valores retornados neste programa, você provavelmente desejará fazer muito mais nos seus programas.

### Exemplo 4.6. Preâmbulo

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

static int server_add_cb(void *data, int type, void *ev);
static int server_del_cb(void *data, int type, void *ev);
static int server_data_cb(void *data, int type, void *ev);

struct Data {
    char *data;
    int data_size;
};
```

Se estiver usando apenas o ecore_con, tudo que você precisa sãos os headers `Ecore.h` e `Ecore_Con.h` para funcionar. Temos umas pré-declarações casuais para manter o compilador feliz e uma estrutura de dados simples que iremos usar em diversos lugares.

### Exemplo 4.7. Entendendo a parte de inicialização

```
int
main(int argc, char ** argv)
{
    struct Data gd;
    struct Data *sd;
    Ecore_Con_Server *srv;
    Ecore_Event_Handler *add = NULL, *del = NULL, *data = NULL;

    sd = calloc(1, sizeof(struct Data));
    gd.data = "In the land of the night "
              "the ship of the sun "
              "is drawn by the grateful dead."
              "~ Egyptian book of the Dead";


    ecore_init();
    ecore_con_init();
```

Iniciamos algumas declarações no corpo do main do programa. Quando você chama o método de conexão do ecore_con, ele irá retornar um `Ecore_Con_Server *` com o qual podemos usar para manipular a conexão se assim desejarmos.

Também estou gardando os `Ecore_Event_Handlers` que vou cirar, para eu mesmo poder liberar os recursos de memória no fim do programa.

A variável `gd` é meu dado “global”. Ela será passada para todas chamadas de cada um dos controladores de evento. A variável `sd` é meu dado “local”. Ela será passada para o controlador quando este servidor estiver trabalhando.

Mas, antes de partirmos de fato para a diversão, devemos nos certificar de inicializar o ecore e ecore_con com as chamadas à `ecore_init()` e `ecore_con_init()`. Como mencionei antes, sempre verifique os valores retornados.

### Exemplo 4.8. capturando os eventos

```
    add = ecore_event_handler_add(ECORE_CON_EVENT_SERVER_ADD,
                                  server_add_cb, &gd);
    del = ecore_event_handler_add(ECORE_CON_EVENT_SERVER_DEL,
                                  server_del_cb, &gd);
    data = ecore_event_handler_add(ECORE_CON_EVENT_SERVER_DATA,
                                  server_data_cb, &gd);
```

Se nós queremos de fato receber algum evento por qualquer motivo, precisamos nós mesmos conectá-lo ao sistema de eventos do ecore. Isto é feito por meio de chamadas à `ecore_event_handler_add()`. No caso de conexão para outro servidor os eventos que estamos interessados são:

| | |
|---|---|
| ECORE_CON_EVENT_SERVER_ADD | Chamado quando conectamos com um servidor |
| ECORE_CON_EVENT_SERVER_DEL | Chamado quando disconectamos do servidor |
| ECORE_CON_EVENT_SERVER_DATA | Chamado quando temos dados recebidos |

| | |
|---|---|
| | do servidor |

Para cada uma das chamadas destes controladores, nós passamos a função que irá controlar o evento e o dado global `gd` que criamos acima. Se não tem qualquer dado pra passar, você pode usar um ponteiro `NULL`.

### Exemplo 4.9. conectando

```
    srv = ecore_con_server_connect(ECORE_CON_REMOTE_SYSTEM,
                                   "www.enlightenment.org", 80, sd);
```

Uma vez que tudo está inicializado, podemos criar uma conexão com o servidor. Isto é feito com uma chamada para `ecore_con_server_connect()`. A chamada irá retornar para nós um `Ecore_Con_Server *` que podemos guardar se desejarmos. A chamada de conexão tem quatro parâmetros. O tipo de conexão, o nome do host, a porta e qualquer dado associado com o servidor.

O tipo de conexão é um dos:

| | |
|---|---|
| ECORE_CON_LOCAL_USER | Conectará com o servidor que estiver escutando no socket Unix em `~/.ecore/[name]/[port]` |
| ECORE_CON_LOCAL_SYSTEM | Conectará com o servidor que estiver escutando no socket Unix em `/tmp/.ecore_service\|[name]\|[port]` |
| ECORE_CON_REMOTE_SYSTEM | Conectará com o servidor que estiver escutando na porta TCP `[name]:[port]` |
| ECORE_CON_USE_SSL | Se a biblioteca foi compilada com suporte à SSL, a conexão será instruída à usar encriptação SSL na transmissão |

O último parâmetro é o dado que é específico (per-server) para cada servidor conectado que queremos passar para cada controlador de evento. Se você não tem este dado "per-server", então pode passar tranquilamente `NULL` como parâmetro.

### Exemplo 4.10. vai speed racer

```
    ecore_main_loop_begin();

    ecore_event_handler_del(add);
    ecore_event_handler_del(del);
    ecore_event_handler_del(data);

    ecore_con_shutdown();
    ecore_shutdown();
    return 0;
}
```

Agora que estamos conectado, nós iniciamos o loop principal de evento com a `ecore_main_loop_begin()`. Co-

mo sou um bom programador, também removo meus controladores de eventos quando saímos do loop principal com a `ecore_event_handler_del()`. Depois disto, é muito importante terminar tudo que inicializamos, `ecore_con_shutdown()` e `ecore_shutdown()` fará esta mágica para nós.

Claro, você pode criar sempre suas conexões depois que o loop principal de eventos foi inicializado, tudo funcionará do mesmo modo (você só precisa registrar os controladores de evento uma vez). Só estou fazendo isto antes porque é mais fácil para um exemplo.

## Exemplo 4.11. adicionado

```
static int
server_add_cb(void *data, int type, void *ev)
{
    Ecore_Con_Event_Server_Add *e;
    struct Data *sd;
    struct Data *gd;
    char buf[1024];

    e = ev;
    gd = data;
    sd = ecore_con_server_data_get(e->server);

    printf("Connected to server ...\n");

    snprintf(buf, 1024, "GET http://www.enlightenment.org/"
                        "pages/enlightenment.html HTTP/1.0\r\n");
    ecore_con_server_send(e->server, buf, strlen(buf));

    snprintf(buf, 1024, "\r\n");
    ecore_con_server_send(e->server, buf, strlen(buf));

    return 1;
}
```

Então agora, a controladora para `ECORE_CON_EVENT_SERVER_ADD`, que neste caso é a `server_add_cb`, irá disparar quando tivermos estabelecido uma conexão com um servidor.

O vento add dará informações sobre este evento por intermédio da estrutura `Ecore_Con_Event_Server_Add`. A princípio, estaremos interessados no membro `server` que é o handler do servidor que foi retornado na chamada de conexão.

O dado global que passamos no controlador de evento será devolvido no parâmetro `void *data` e o dado "per-server" que passamos na chamada de conexão pode ser obtido com uma chamada para a `ecore_con_server_data_get()`.

Agora que estamos conectado com o servidor, podemos enviar um pedido de um documento, que neste caso estou enviando um pedido HTTP. O dado é enviado para o servidor chamando a `ecore_con_server_send()` e passando como parâmetros o servidor, o dado e o tamanho do dado.

Certifique-se de retornar 1 para cada um dos seus controladores, pois se você retornar 0 o Ecore removerá o controlador e de evento da lista de controladores disponíveis.

## Exemplo 4.12. removido

```
static int
server_del_cb(void *data, int type, void *ev)
```

```
{
    Ecore_Con_Event_Server_Del *e;
    struct Data *sd;
    struct Data *gd;

    e = ev;
    gd = data;
    sd = ecore_con_server_data_get(e->server);

    ecore_con_server_del(e->server);

    printf("%s\n\n", gd->data);
    if (sd->data) {
        printf("%s\n", sd->data);
        free(sd->data);
    }

    ecore_main_loop_quit();
    return 1;
}
```

Vamos agora para o controlador `ECORE_CON_EVENT_SERVER_DEL`. Ele funciona de forma similar ao controlador add, mas o tipo de evento passado é um `Ecore_Con_Event_Server_Del`.

Agora, o callback del será disparado quando formos disconectado do servidor remoto. Isto significa que é nossa responsabilidade remover da memoria o servidor. Isto é feito com a chamada à `ecore_con_server_del()`. Nós, é claro, não precismos fazer isto aqui, podemos fazer em qualquer momento quando nos for conveniente.

No caso deste exemplo, já que o servidor fechará a conexão uma vez que a pagina web foi retornada, estou fazendo o processamento dos dados no controlador del. Como não quero continuar após pegar a pagina chamo `ecore_main_loop_quit()` para parar o loop principal de evento.

## Exemplo 4.13. dado

```
static int
server_data_cb(void *data, int type, void *ev)
{
    Ecore_Con_Event_Server_Data *e;
    struct Data *sd;
    struct Data *gd;

    e = ev;
    gd = data;
    sd = ecore_con_server_data_get(e->server);

    sd->data = realloc(sd->data, sd->data_size + e->size + 1);
    memcpy(sd->data + sd->data_size, e->data, e->size);

    sd->data_size += e->size;
    sd->data[sd->data_size] = '\0';

    return 1;
}
```

A `ECORE_CON_EVENT_SERVER_DATA` é também similiar aos callbacks add e del. A estrutua de evento desta vez é uma `Ecore_Con_Event_Server_Data` que apresenta para nós dois novos membros interessantes, que são: `size` e `data`. Que nos fornecem o dado recebido do servidor e o tamanho deste dado. Este dado *não* é terminado com NULL, portanto certifique-se de usar o parâmetro `size`.

No meu caso, só estou guardando o dado na minha estrutura "per-server" enquanto pego a desconexão. Você pode, é

claro, fazer qualquer processamento que deseja aqui.

### Exemplo 4.14. compilação

```
zero@oberon [ecore_con] -> gcc -o srv main.c `ecore-config --cflags --libs`
```

Uma vez feito tudo isto, a compilação é a parte simples. O programa pode ser executado com o simples comando

```
./srv
```

e deverá imprimir na tela a pagina web selecionada.

Ecore_Con torna fácil trabalhar com quaisquer tipos de servidores remotos, seja HTTP, IRC ou qualquer um customizado, todas as funcionalidade estão encapsuladas e controladas muito bem atravéz do loop de evento Ecore.

# Receita: Introdução a Ecore Ipc

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

A biblioteca Ecore_Ipc provê um "wrapper" robusto e eficiente para o módulo Ecore_Con. Ecore_Ipc lhe permite configurar as comunições do seu servidor e controlar todos os aspectos internos chatos necessários. Este receita dará um exemplo simples de um cliente Ecore e um servidor Ecore.

Quando trabalhamos com Ecore_Ipc, escrevendo uma aplicação cliente ou um servidor, um objeto Ecore_Ipc_Server será criado. Porque em qualquer caso há um servidor sendo manipulado, para ser inicializado ou para se comunicar. Depois disto, todo o resto é fácil.

### Exemplo 4.15. Cliente Ecore_Ipc: preâmbulo

```
#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);
```

O arquivo Ecore.h é incluido para que possamos ter acesso ao tipo de sinal de saída. As funções serão explicadas mais tarde quando seus callbacks são conectados.

### Exemplo 4.16. Cliente Ecore_Ipc: início do 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, sempre que estamos escrevendo uma aplicação cliente, também usamos um objeto Ecore_Ipc_Server. Usar Ecore_Ipc requer a inicialização de Ecore. Isto se faz com uma simples chamada à `ecore_init`. Ecore_Ipc é então iniciado com uma chamada à `ecore_ipc_init`. Caso alguma delas devolver 0, é tomada a ação apropriada para desfazer qualquer inicialização feita até este momento. O callback ECORE_EVENT_SIGNAL_EXIT é conectado de modo que podemos sair corretamente se for necessário.

### Exemplo 4.17. Cliente Ecore_Ipc: main criando o 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);
```

Neste exemplo criamos uma conexão remota com o servidor chamado "localhost" na porta 9999. Isto é feito com o método `ecore_ipc_server_connect`. O primeiro parâmetro é o tipo de conexão que se deseja fazer, podendo ser: ECORE_IPC_REMOTE_SYSTEM, ECORE_IPC_LOCAL_SYSTEM ou ECORE_IPC_LOCAL_USER. Se OpenSSL estava disponível quando Ecore_Ipc foi compilado, ECORE_IPC_USE_SSL pode ser combinado com o tipo de conexão por meio de um OR para criar uma conexão SSL.

As três chamadas à `ecore_event_handler_add` configura os callbacks para os diferentes tipos de eventos que receberemos do servidor. Um servidor foi adicionado, um servidor foi eliminado ou o servidor nos enviou dados.

### Exemplo 4.18. 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 o propósito deste exemplo, o cliente quando iniciar enviará uma mensagem para o servidor do qual responderá. A mensagem do cliente é enviando com o comando `ecore_ipc_server_send`. Este comando, `ecore_ipc_server_send`, pega o servidor IPC, os opcodes maior e menor, os parâmetros ref e ref_to, o response, os dados e o tamanho dos dados. Este parâmetros, exceto o servidor IPC, são definidos pelo cliente e podem referir-se a qualquer coisa que se deseja. Isto possibilita dar máxima flexibilidade na criação de aplicações IPC cliente/servidor.

Depois de enviar a mensagem entramos no lopp principal ecore e esperamos os eventos. Se sairmos do loop, apaga-

mos o objeto servidor, finalizamos Ecore_Ipc chamando a `ecore_ipc_shutdown`, e finalizamos o ecore com `ecore_shutdown`.

### Exemplo 4.19. Cliente Ecore_Ipc: sig_exit_cb

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

A `sig_exit_cb` simplesmente diz ao ecore para sair do loop principal. Ela não é estritamente necessária, se a única coisa que está sendo feito é chamar a `ecore_main_loop_quit()`, o Ecore controlará isto por sí próprio se não houver um controlador. Mas isto mostra como um controlador pode ser criado se necessitar de um na sua aplicação.

### Exemplo 4.20. Cliente Ecore_Ipc: os 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;
}
```

Este três callbacks, `handler_server_add`, `handler_server_del`, e `handler_server_data`, são o corpo do cliente controlando todos os eventos relacionados com o servidor que estamos conectados. Cada um destes callbacks tem uma estrutura de evento associada, Ecore_Ipc_Event_Server_Add, Ecore_Ipc_Event_Server_Del e Ecore_Ipc_Event_Server_Data, contendo informação sobre o próprio evento.

Quando conectamos pela primeira vez com o servidor, a função callback `handler_server_add` será executada permitindo fazer qualquer inicialização.

Se o servidor quebrar a conexão, `handler_server_del` será executada permitindo qualquer limpeza requerida.

Quando o servidor envia dados para o cliente, `handler_server_data` será executada. Que neste exemplo apenas imprime alguma informação sobre a própria mensagem e seu corpo.

E este é a parte cliente. O código é bastante simples graças as abstrações providas pelo Ecore.

### Exemplo 4.21. Servidor Ecore_Ipc : preâmbulo

```
#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 no cliente, o header Ecore.h é incluído para ter acesso ao sinal exit. O header Ecore_Ipc.h é requerido para aplicações que fazem uso a biblioteca Ecore_Ipc. Cada controlador de sinal será explicado no seu próprio código.

### Exemplo 4.22. Servidor Ecore_Ipc: início 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);
```

Aqui é igual a inicialização do cliente.

### Exemplo 4.23. Servidor Ecore_Ipc: criando o 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);
```

Diferente do cliente, para o servidor nós adicionamos um listener para a porta 999 na máquina "localhost" por meio da chamada a `ecore_ipc_server_add`. Isto criará e devolverá o objeto servidor. Então, conectamos os controladores de eventos necessários, a diferência com o cliente é que aqui queremos os eventos CLIENT no lugar dos eventos SERVER.

### Exemplo 4.24. Servidor Ecore_Ipc: final de main

```
    ecore_main_loop_begin();

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

```
}
```

Novamente isto é idêntico ao cliente, menos o envio de dados ao servidor.

### Exemplo 4.25. Servidor Ecore_Ipc: callback sig_exit

`sig_exit_cb` novamente igual ao que foi visto no cliente.

### Exemplo 4.26. Servidor Ecore_Ipc: os 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;
}
```

Os callbacks de evento são similiares aos vistos na aplicação cliente. A principal diferência é que estes são eventos _Client_ em vez de eventos _Server_.

O callback add é quando um cliente se coencta ao nosso servidor, o callback del, em oposição, é quando o cliente se desconecta. O callback data é para quando um cliente envia dados para o servidor.

Ao final do callback `handler_client_data` fazemos uma chamada a `ecore_ipc_client_send`. Isto envia os dados ao cliente. Como no envio de dados ao servidor, os parâmetros são: o cliente que receberá os dados, os opcodes maior e menor, ref, ref_to, response, os dados e o tamanho dos dados.

### Exemplo 4.27. Ecore_Ipc: compilação

```
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 em outras aplicações ecore, é muito fácil compilar uma aplicação Ecore_Ipc. Se sua biblioteca ecore foi compilada com suporte a Ecore_Ipc, simplesmente invocando o comando 'ecore-config --cflags --libs' adicionará todos os caminhos das bibliotecas requeridas e as informações de linkagem.

Como foi visto neste exemplo, Ecore_Ipc é uma biblioteca fácil de usar para criar aplicações cliente/servidor.

# Receita: Temporizadores Ecore

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

Se você precisa que um callback seja disparado num determinado momento com a possibilidade de repetir o temporizador continuamente, então `Ecore_Timer` é o que você procura.

### Exemplo 4.28. 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;
}
```

A criação dos temporizadores é simples bastando chamar `ecore_timer_add()`. Isto retornará uma estrutura `Ecore_Timer` ou NULL se a chamada falhar. Neste caso eu estou ignorando o valor de retorno. Os três parâmetros são:

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

O parâmetro timeout dá o número de segundos que expirará o temporizador. No caso do exemplo, demos 1 segundo e 0.5 segundos respectivamente. A função callback é a que será executada quando o temporizador expirar e user_data é qualquer dado para ser passado para a função callback.

As funções callback tem a mesma assinatura `int callback(void *data)`. O valor de retorno do temporizador deve ser 0 ou 1. Se você retornar 0 o temporizador expirará e *não* será executado novamente. Se você retornar 1, o temporizador será re-agendado para se executar novamanete após o fim do tempo dado pelo parâmetro timeout. Isto permite que você ative ou continue com o temporizador do jeito que necessitar seu programa.

Se você tem um temporizador que deseja remover em um determinado momento basta chamar `ecore_timer_del(Ecore_Timer *)`. Se a eliminação for bem sucedida o ponteiro será retornado, caso contrário será retornado NULL. Depois disto a estrutura `Ecore_Timer` será invalidada e você não poderá mais usar novamente no seu programa.

Compilar o exemplo é bem simples:

### Exemplo 4.29. Compilação

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

Se você executar o programa deverá ver uma série de '1's e '2's na tela com duas vezes mais '2's que '1's.

Os temporizadores `Ecore_Timer` são fáceis de usar provendo um potente mecanismo de temporização para seus programas.

# Receita: Adicionando eventos Ecore

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

Se você sempre precisa fazer seus próprios eventos você pode facilmente conecta-los no sistema de eventos Ecore. Ele dá a você a capacidade de adicionar eventos dentro de uma fila de eventos e ter o Ecore para remete-lo para outra parte da aplicação.

O seguinte programa cria um simples evento e temporizador. Quando o temporizador dispara ele irá adicionar seu novo evento dentro de uma fila de eventos. Então seu evento irá imprimir uma mensagem e terminará a aplicação

### Exemplo 4.30. Exemplo de evento Ecore

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

static int timer_cb(void *data);
static int event_cb(void *data, int type, void *ev);
static void event_free(void *data, void *ev);

int MY_EVENT_TYPE = 0;

typedef struct Event_Struct Event_Struct;
struct Event_Struct {
    char *name;
};

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

```
    MY_EVENT_TYPE = ecore_event_type_new();
    ecore_event_handler_add(MY_EVENT_TYPE, event_cb, NULL);
    ecore_timer_add(1, timer_cb, NULL);

    ecore_main_loop_begin();
    ecore_shutdown();
    return 0;
}

static int
timer_cb(void *data)
{
    Event_Struct *e;
    Ecore_Event *event;

    e = malloc(sizeof(Event_Struct));
    e->name = strdup("ned");

    event = ecore_event_add(MY_EVENT_TYPE, e, event_free, NULL);
    return 0;
}

static int
event_cb(void *data, int type, void *ev)
{
    Event_Struct *e;

    e = ev;
    printf("Got event %s\n", e->name);
    ecore_main_loop_quit();
    return 1;
}

static void
event_free(void *data, void *ev)
{
    Event_Struct *e;

    e = ev;
    free(e->name);
    free(e);
}
```

Cada evento tem um id de evento associado. Este id é simplesmente um valor inteiro que é fornecido quando se chama `ecore_event_type_new()`. Uma vez que nós temos o id de evento podemos usá-lo nas chamadas `ecore_event_handler_add()`. Isto é tudo o que precisa para criar o evento.

Ecore nos dá a capacidade de passar uma estrutura de evento para nosso evento. Só que você precisa ter cuidado, se você não especificar uma função para liberar a memória (free function) usada pela estrutura, a ecore irá usar uma função genérica que apenas chama `free` pro valor. Então, não coloque nada na estrutura que você precise sem tomar cuidado. (Ou prepare-se para uma enxurrada de SEGV's muito estranhas no seu programa)

Neste exemplo nós criamos uma simples `Event_Struct` que é passado. A chamada que de fato cria o evento é `ecore_event_add()`. Que recebe o id de evento, o dado de evento, a "free function" e qualquer dado para passar para a "free function".

Como você pode ver, nós passamos nossa `Event_Struct` como um dado de evento e informamos a função `event_free` como a "free function" que se encarregará de limpar da memória a estrutura.

E é isto. Você pode compilar como descrito abaixo e tudo deve funcionar.

## Exemplo 4.31.

```
zero@oberon [ecore_event] -> gcc -o ev main.c `ecore-config --cflags --libs`
```

Como visto, é realmente muito fácil extender o Ecore com seus próprios eventos. O sistema foi configurado para permitir ser extendido como desejado.

# Capítulo 5. EDB & EET

EDB é uma biblioteca de conveniência de base de dados envolvendo o Berkeley DB 2.7.7 da Sleepycat Software. Seu propósito é fazer de forma fácil, rápido, eficente e portável o acesso as informações da base de dados.

EET é uma pequena biblioteca desenhada para escrever pedaçoes arbitrários de dados em um arquivo, opcionalmente compactá-los (muito semelhante a um arquivo zip) e permitir uma leitura veloz para acesso aleatório mais tarde. EET não faz zip, dado que um zip tem mais complexidade que o necessário, e foi mais simples implementar isto uma vez aqui.

EDB fornece um método excelente de armazenar e recuperar informações de configuração de aplicação, contudo pode ser usado de uma maneira mais extensíva que esta. Ebits, o predecessor da Edje, usava EDB como container para temas Ebit antes da EET. Uma Edb consiste em uma série de pares valor/chave, que pode consistir numa variedade de tipos de dados, incluindo inteiros, valores de ponto flutuante, strings e dados binários. A API simplificada provê funções simples, completas e unificadas para gerenciar e acessar seu banco de dados.

Em adição à biblioteca, há disponível uma variedade de ferramentas para acessar e modificar seus EDBs. O edb_ed fornece uma simples interface de linha de comando que pode ser fácilmente usada em scripts, especialmente útil para uso com o GNU autotools. O edb_vt_ed fornece uma inferface ncurses fácil de usar. Finalmente, edb_gtk_ed fornece uma elegante e fácil interface GUI, especialmente útil para o usuário final editar os dados contidos nos EDBs.

Eet é extremamente rápida, pequena e simples. Os arquivos Eet podem ser pequenos e altamente compactados, tornando-os ótimos para enviar pela internet sem ter que arquivar, compactar ou descompcatar, e instalá-los. Permitem leituras velozes de acesso aleatórios como um ráio uma vez criados, fazendo-os perfeitos para armazenar dados que são escritos uma vez (ou raramente) e lidos muitas vezes, mas o programa não quer ter que lê-lo todas as vezes.

Também pode codificar e descodificar estruturas de dados na memória, bem como dados de imagem para gravar em arquivos Eet ou enviar pela rede à outras máquinas, ou apenas escrever arquivos arbitrário no sistema. Todos os dados codificados são independente da plataforma podendo ser lidos e escritos em qualquer arquitetura.

# Receita: Criando arquivos EDBs pelo shell

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

Muitas vezes é desejado criar arqiuvos EDB por intermédio de um simples script shell. Para então ser parte do processo de construção.

Isto pode ser facilmente implementado usando o programa **edb_ed**. **edb_ed** é uma interface muito simples para EDB, permitindo criar/editar/apagar pares chave/valor dentro da base de dados EDB.

### Exemplo 5.1. Arquivo de escript shell 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
```

Se o arquivo de output não existe na primeira chamada ao comando `add`, então **edb_ed** criará o arquivo e fará qualquer inicialização necessária. A `add` é usada para adicionar entradas na base de dados. O primeiro parâmetro depois de add é a chave que os dados serão inseridos na base de dados. Esta chave será usada no futuro para atualizar os da-

dos pela sua aplicação. O próximo parâmetro é o tipo do dado que será adicionado. Que pode ser:

- int
- str
- float
- data

O último parâmetro é o valor que será associado com esta chave.

Usando o **edb_ed** você pode criar/editar/visualizar qualquer arquivo EDB requerido pela sua aplicação de maneira fácil e rápida.

# Recetta: Introdução ao EDB

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

EDB fornece um potente backend de base de dados para usar nas suas aplicações. Esta receita é uma simples introdução que abrirá um banco de dados, escreverá várias chaves e depois lerá os dados.

### Exemplo 5.2. Introdução ao 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 o EDB você deve incluir o <Edb.h> no seu arquivo para ter acesso a API. A parte inicial do programa é bastante padrão, tenho a tendência de digitar errado enquanto defino as diferentes chaves que irei usar. Enquanto temos o nome do arquivo tentaremos abrir/criar a base de dados.

A base de dados será aberta ou, se não existir, será criada com a chamada à `e_db_open()` que devolverá NULL se ocorreu um erro.

Uma vez aberta, podemos escrever nossos valores. Isto é feito por meio das três chamadas: `e_db_int_set()`, `e_db_float_set()` e `e_db_str_set()`. Você também pode inserir dados genéricos dentro do arquivo .db com `e_db_data_set()`.

Junto com dados normais, você pode guardar metadados sobre a base de dados no próprio arquivo. Eles não podem ser recuperados com os métodos tradicionais get/set. Estas propriedade são tratadas com `e_db_property_set()`

Cada um dos métodos de atribuição de tipo (get/set) usa três parâmetros:

- E_DB_File *db
- char *key
- value

O parâmetro `value` é do tipo correspondente do metódo, int, float, char * ou void * para _int_set, _float_set, _str_set e _data_set respectivamente.

Uma vez que os valores estão na base de dados, eles podem ser recuperados com os métodos de acesso. Cada um destes métodos usa três parâmetros e retorna um int. O valor de retorno é 1 em caso de sucesso e 0 caso contrário.

Como nos métodos de atribuição, os parâmetros dos métodos de acesso são o db, a chave e um ponteiro para o valor recuperado.

Quando terminamos com a base de dados podemos fecha-la com uma chamada à `e_db_close()`. A chamada à `e_db_close()` não dá garantia que os dados foram escritos no disco, para isto chamamos `e_db_flush()` que escreverá o conteúdo da base de dados no disco.

### Exemplo 5.3. Compilando

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

Se você executar o programa deverá ver os valores na tela, depois da execução haverá um arquivo .db com o nome que você especificou. Você pode dar uma olhada no arquivo .db com o **edb_gtk_ed** e ver os valores entrados.

# Receita: Recuperação de chave EDB

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

A API EDB faz da recuperação de todas as chaves disponíveis na base de dados uma tarefa simples. Estas chaves podem então ser usadas para determinar os tipos de objetos na base de dados, ou simplesmente para recuperar o objeto.

### Exemplo 5.4. recuperação de chaves 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;
}
```

Recuperação de chaves se faz simplesmente por meio da chamada à `e_db_dump_key_list()`. Que devolverá um array do tipo string (char**) contendo as chaves. Estas strings, e o próprio array, devem ser removidos da memória pela aplicação. `e_db_dump_key_list()` devolverá também o número de chaves contidas no array pelo parâmetro `num_keys`.

Você notará que não precisamos abrir a base de dados para chamar a `e_db_dump_key_list()`. Esta função trabalha com o arquivo em si no lugar de um objeto db.

### Exemplo 5.5. Compilação

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

Executando o programa deverá produzir uma lista de todas as chaves obtidas da base de dados. Isto pode ser verificado olhando a base de dados por meio de uma ferramente externa como **edb_gtk_ed**.

# Capítulo 6. Esmart

Esmart fornece uma variedade de objetos inteligentes EVAS que dá uma potência significativa em suas aplicações baseadas no EVAS e EFL.

# Receita: Indrodução ao Esmart Trans

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

Transparência está se tornando cada vez mais uma peculiaridade comum nas aplicações. Para este fim, o objeto Esmart_Trans foi criado. Este objeto fará todo o trabalho difícil de se produzir um fundo transparente para seus programa.

Esmart_Trans faz a integração do fundo transparente na sua aplicação muito fácil. Você precisa criar o objeto trans, e então certificar-se de atualiza-lo conforme a janela vai sendo movida ou redimensionada.

### Exemplo 6.1. Declarações e Inclusões

```
#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 aplicação que usa um objeto Esmart_Trans precisa do Ecore, Ecore_Evas, e os arquivos headers das Esmart/Esmart_Trans. A próximas quatro declarações são callbacks do ecore para eventos para nossa janela, saída, remoção, redimencionamento e movimento respectivamente. As últimas declarações são funções utilitárias que são usadas no exemplo e não necessitam estar no seu programa.

### Exemplo 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;
}
```

A rotina principal para este programa é bem simples. Ecore e Ecore_Evas são ambas inicializadas com as devidas checagem de erros. Então criamos a gui e iniciamos o loop principal de evento ecore. Quando o ecore sai encerramos tudo e retornamos.

### Exemplo 6.3. callbacks de saída e remoção

```
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();
}
```

Os callbacks de saída e remoção são os callbacks ecore genéricos. O callback de saída não é estritamente necessário, já que o Ecore irá chamar a `ecore_main_loop_quit()` se não houver nenhum controlador registrado, mas está incluído pra mostrar como isto é feito.

### Exemplo 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);
}
```

A rotina `_freshen_trans` é uma função de ajuda para atualizar a imagem que a transpaência se mostra. Está será chamada quando precisamos atualizar nossa imagem da qual atualmente está em baixo. A função captura o tamanho atual do ecore_evas, e então adquire o objeto com o nome "bg" ( este nome é o mesmo que damos a nossa trans quando a criamos ). Então, desde que o objeto exista, pedimos para a esmart atualizar a imagem sendo mostrada.

### Exemplo 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);
}
```

Quando a janela for redimensionada precisamos atualizar nosso evas para o tamanho correto e então atualizar o objeto trans para mostrar a parte do background. Capturamos o tamanho atual da janela com `ecore_evas_geometry_get` e também o tamanho máximo e mínimo da janela. Com o tamanho desejado, o tamanho basea-se entre os limites mínimo e máximo da janela, capturamos o objeto "bg" ( novamente o mesmo do título ) e o redimensionamos. Uma vez feito o redimensionamento, chamamos a rotina `_freshen_trans` para atualizar a imagem mostrada no bg.

### Exemplo 6.6. move_cb

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

Quando a janela é movimentada precisamos autalizar a imagem mostrada como transparência.

### Exemplo 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);
```

A primeira parte da make_gui se encarrega de iniciar a ecore e a ecore_evas. Primeiro o callback de saída é conecta-

do ao ECORE_EVENT_SIGNAL_EXIT, então o objeto Ecore_Evas é criado com o motor do X11. Se inicia o título da janela e conectamos os callbacks escrios acima, delete, resize e move. Finalmente capturamos a evas para o Ecore_Evas criado.

### Exemplo 6.8. Criando o 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);
}
```

Uma vez que tudo esteja iniciado podemos criar o objeto trans. O trans é criado no evas devolvido pela `ecore_evas_get`. Esta criação inicial é feita mediante a chamada à `esmart_trans_x11_new`. Uma vez com o objeto, o movemos de maneira que ele incie na posição( 0, 0 ) ( o canto superior esquerdo ), ajustamos a camada para -5 e chamamos o objeto "bg" ( como usamos acima ). Então capturamos o tamanho atual do ecore_evas e o usamos para redimensionar o objeto trans para o tamanho da janela. Quando tudo estiver redimensionado mostramos o trans e o ecore_evas. Como passo final, atualizamos a imagem na transparência da imagem que está atualmente em baixo da janela da forma como está.

### Exemplo 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 o programa acima precisamos incluir a informação de biblioteca para ecore, ecore_evas e esmart. Isto é feito por meio dos escripts -config para cada biblioteca. Estes escripts -config sabem onde estão cada um dos includes e das bibliotecas iniciando os caminhos (paths) de include e os parâmetros de linkagem apropriados para a compilação.

# Receta: Introdução ao Container Esmart

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

Geralmente ao desenhar a UI de uma aplicação há um desejo de agrupar os elementos comuns juntos e fazer com que seu layout dependa um do outro. Para este fim a biblioteca de Container Esmart foi criada. Ela foi projetada para contolar a parte difícil de layout, e em casos onde ela não faz o que você precisa, as porções de layout do container são extensíveis e modificáveis.

Esta receita dará a base para usar um container Esmart. O produto final é um programa que te permitirá ver algumas das diferentes combinações de layout do container default. O layout será feito pelo Edje com callbacks para o pro-

grama mudar o layout do container e para informar se o usuário clicou em um elemento do container.

### Exemplo 6.10. Includes e declarações

```
#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 nos outros programas EFL precisamos incluir Ecore, Ecore_Evas, Edje e como este é um exemplo de container, o cabeçalho do Esmart/Esmart_Container. Getopt será usado para permiir algum processo de linha de comando.

Depois vem os protótipos de funções que serão descritos mais tarde quando chegarmos às suas respectivas implementações. Depois, algumas variáveis globais que serão usadas durante o programa. O array str_list é o conteúdo a ser ordenado no container.

### Exemplo 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);
}
```

O inicio da função main pega as opções de linha comando e inicia a janela principal. Como você pode ver, solicitamos um tema para mostrar. Isto pode ser mais inteligente pocurando no diretório default de instalação e no diretório de aplicações do usuário, mas este exemplo prefere o modo mais fácil e força que o tema seja passado como argumento de linha de comando.

## Exemplo 6.12. Inicialização

```
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();
```

Depois de receber os argumentos de linha de comando, continuamos por inicializar as bibliotecas requeridas, Ecore, Ecore_Evas e Edje. Tomamos o passo adicional de iniciar o frame rate do Edje.

Uma vez completada a inicialização, criamos a GUI inicial para a aplicação. Separei a construção do container em uma função separada para deixar o código do container mais fácil de localizar no exemplo.

Quando tudo está criado, chamamos a `ecore_main_loop_begin` e esperamos que ocorram os eventos.

## Exemplo 6.13. Finalização

```
    edje_shutdown();

EXIT_ECORE_EVAS:
    ecore_evas_shutdown();

EXIT_ECORE:
    ecore_shutdown();

EXIT:
    return ret;
}
```

A rotina de finalização habitual nos faz bons programadores e terminamos tudo o que inicializamos.

### Exemplo 6.14. callbacks de janela

```
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);
        }
    }
}
```

No passo seguinte iniciamos alguns callbacks genéricos para ser usados pela UI. Estes serão os callbacks de sair, destruir e redimensionar. Novamente, as funções habituais EFL. Apesar que o callback de saída não é estritamente necessário já que Ecore chamará a `ecore_main_loop_quit()` se não houver nenhum controlador registrado para este callback.

### Exemplo 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);
}
```

A GUI consiste do Ecore_Evas que contém o próprio canvas, e o Edje que usaremos para controlar nosso layout. A função `make_gui` inicia os callbacks definidos mais acima e cria o Ecore_Evas.

Uma vez definido o Evas e os callbacks, criamos o objeto Edje que definirá nosso layout. A chamada à `edje_object_add` é usada para criar o objeto no Evas, e uma vez que isto é feito, pegamos o tema passado pelo usuário e configuramos o Edje para usar este tema, o parâmetro "container_ex" é o nome do grupo dentro do EET que vamos usar.

Uma vez que o arquivo de tema foi passado para o Edje, usamos os valores no arquivo de tema para inciar os limites do tamanho da aplicação, e mostramos o Edje. Então iniciamos os callbacks no Edje e mostramos o Ecore_Evas.

## Exemplo 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);
}
```

O programa terá dois callbacks principais conectados ao Edje, um para o sinal de clique com o botão esquerdo do mouse e outro para o sinal do botão direito. Estes serão usados para mudar a direção/alinhamento do container. O segundo e o terceiro parametros dos callbacks precisam bater com os dados emitidos com o sinal do Edje, isto será visto mais tarde quando vermos o arquivo EDC. O terceiro paramêtro é a função para chamar, e o ultimo, qualquer dado que desejamos passar pro callback.

### Exemplo 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);
```

A função `container_build` criará um container e inicializará nossos elementos de dados no container. A criação é suficientement fácil com uma chamda à `esmart_container_new` devolvendo o Evas_Object que é o container. Uma vez criado podemos dar um nome ao container para fazer a referência mais fácil.

O próximo passo, iniciamos a direção, que é CONTAINER_DIRECTION_VERTICAL ou CONTAINER_DIRECTION_HORIZONTAL [ou neste caso, um int passado pela linha de comando que refere-se a uma das direções sendo 1 e 0 respectivamente]. A direção informa ao container de que maneira os elementos serão desenhados.

Depois da direção iniciamos o alinhamento do container. O alinhamento informa ao container de onde desenhar os elementos. Os valores possíveis são: ONTAINER_ALIGN_CENTER, CONTAINER_ALIGN_LEFT, CONTAINER_ALIGN_RIGHT, CONTAINER_ALIGN_TOP e CONTAINER_ALIGN_BOTTOM. Com o layout default, direita e esquerda se aplicam apenas a um container vertical, e acima e abaixo se aplicam a um container horizontal, enquanto centro se aplica a ambos.

Se desejarmos usar um esquema de layout diferente do default, poderemos fazer por uma chamada a `esmart_container_layout_plugin_set(container, "name")` onde "name" é o nome do plugin à usar. A configuração default é o container nomeado "default".

Uma vez inicializado as direções e o alinhamento, o espaçamento e o preenchimento do container são especificados. O preenchimento especifica o espaço em volta do container passando quatro parâmetros: left, right, top e buttom. O parâmetro de espaçamento especifica o espaço entre os elementos no container.

Então continuamos e iniciamos a política de preenchimento do container. Isto especifica como os elementos são posicionados para preencher o espaço no container. Os valores possíveis são: CONTAINER_FILL_POLICY_NONE, CONTAINER_FILL_POLICY_KEEP_ASPECT, CONTAINER_FILL_POLICY_FILL_X, CONTAINER_FILL_POLICY_FILL_Y, CONTAINER_FILL_POLICY_FILL e CONTAINER_FILL_POLICY_HOMOGENOUS.

Uma vez que o container é completamente especificado, iniciamos a camada de containers e "tragamos" o container para dentro do ejde e a parte chamada "container".

### Exemplo 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);
}
```

Agora que temos um container, podemos adicionar alguns elemenos para mostrar. Cada uma das entradas no array str_list definido no inicio do programa será adicionada no container como uma parte de texto.

Para cada elemento criamos uma novo objeto Edje no Evas. Então necessitamos saber o nome do arquivo de tema usado para criar nosso Edje principal, chamamos `edje_object_file_get` que iniciará o arquivo edje para o valor informado.

Então tentamos iniciar o grupo chamado "element" no elemento novamente criado. Se isto falhar imprimimos um erro e deletamos o objeto.

A medida qe vamos encontrando o grupo "element" podemos capturar a parte para do nosso elemento chamada "element.value". Se esta parte existir, ajustamos o valor de texto da parte para nossa string atual e mostramos a parte.

Um callback é criado por intermédio da `edje_object_signal_callback_add` e vinculado ao novo elemento. Este será chamado se o sinal "item_selected" é enviado pelo Edje. O valor i_ptr mostra como se pode vincular dados ao elemento, quando o usuário clicar em um elemento seu número será impresso no console.

Uma vez criado o elemento adicionamos-o ao container.

Por fim, o container é mostrado e fazemos algum trabalho extra para mostrar informação sobre o container no cabeçalho por meio da chamada `_show_text`.

## Exemplo 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");
}
```

A rotina `_set_text` pega a direção e o alinhamento atual do container e coloca algum texto no cabeçalho do programa. Isto é apenas uma simples comunicação com o usuário da configuração do container atual.

## Exemplo 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);
}
```

Quando o usuário clica com o botão esquerdo do mouse este callback irá ser executado. Nós pegamos a informação de direção atual do container e mudamos para a outra direção (ex. horizontal se torna vertical e vice-versa). Também reiniciamos o alinhamento se não estamos atualmente alinhado no centro para certificarmos que tudo é válido para a nova direção. O texto no cabeçalho é atualizado.

## Exemplo 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));
}
```

O calback do clique do botão direito intercalará entre os alinhamentos disponíveis por uma direção dada quando o usuário clicar com o botão direito do mouse.

### Exemplo 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 o callback `_item_selected` será executado quando o usuário clicar no botão central do mouse sobre o item do container. O dado conterá o número para este elemento na rotina criada acima.

Este é o fim do código para a aplicação, depois vem a EDC requerida para que tudo seja mostado e funcione corretamente.

### Exemplo 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 arquivo EDC espera ter a font Vera incorporada dentro dele, como é definido pela seção de fontes no inicio. Isto quer dizer que quando você compila o edc você necessita do arquivo Vera.ttf no diretório corrente ou dar ao edje_cc a flag -fd e especificar o diretório para a fonte.

Depois que as fontes são definidas, as coleções principais são definidas. A primeira coleção é a porção principal da própria aplicação, o grupo "container_ex". Este grupo especifica a janela principal da aplicação. Como tal ele contêm as partes para o fundo, o cabeçalho e o texto de cabeçalho. Estas partes são bem padrão com algum (mínimo) alinhamento feito entre elas.

## Exemplo 6.24. A Parte Container

```
        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";
        }
    }
```

```
    }
```

A parte container é então definida. A parte em si é bem siples, apenas posicionada relativamente ao fundo e iniciada para receber eventos do mouse. Depois de definir as partes especificamos os programas para este grupo, do qual há dois. O primeiro programa "left_click" espeficia o que vai acontecer em um evento do primeiro botão do mouse.

A ação é emetir um sinal, os dois parâmetos depois de SIGNAL_EMIT bate com os valores postos no callback no código da aplicação.

Há um callback similar para o terceiro botão do mouse assim como para o primeiro, só que emitindo um sinal levemente diferente.

## Exemplo 6.25. O 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";
            }
        }
    }
}
```

O grupo elemento especifica como cada elemento do container é mostrado. Você notará que os nomes aqui dados batem com os nomes que se procuram no próprio código da aplicação enquanto se criam os elementos.

Há um programa neste grupo que irá emitir um sinal de "item_selected" quando o botão central do mouse é pressionado enquanto estivermos sobre um dos elementos na lista.

Este é o final do código EDC. Para compliar o código da aplicação, um makefile similar ao abaixo pode ser usado.

## Exemplo 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)
```

E para criar o arquivo EET, um simples 'edje_cc default.edc' deve ser suficiente desde que o arquivo Vera.ttf esteja no diretório atual.

Uma vez que você tenha compilado, você precisará fazer

```
./container_ex -t default.edj
```

e tudo deve funcionar bem.

Então é isto, assumindo que tudo saia como planejado, você deverá ter uma simples aplicação em que clicando com o botão esquerdo/direito do mouse moverá o container para diferentes posições da jenela. Enquanto clicando com o botão central do mouse nos elementos, imprimirá o número do elemento.

# Capítulo 7. Epeg y Epsilon

Nesta era moderna de fotografia digital a apresentação de imagens se converte em um problema devido ao grande volume de imagens que são criadas. Diferente do passado, quando um filme era usado equilibradamente, agora geramos centenas ou milhares de imagens por semana. A solução para este problema de apresentação de imagens é "thumbnail", uma imagem em escala reduziada que pode ser indexada em uma tabela ou por uma aplicação e rápidamente scaneada visualmente para encontrar as imagens que deseja. Mas escalonamento de imagem é uma operação muito intensiva, tanto que numa potente máquina Athlon escalonar apenas uma fotografia de tamanho 1600x1200 na resolução requerida leva-se 1 segundo, se tiver 2000 fotografias levará 30 minutos assumindo que não há operação manual num editor de imagens como o Photoshop ou o GIMP. O problema claramente pede uma ferramente que pode escalar imagens com uma grande velocidade e eficiência, com tanto controle quanto possível. A solução está em duas bibliotecas da EFL : Epeg e Epsilon.

Epeg foi escrita por Raster para exatamente controlar o problema acima descrito com sua galeria de imagens em seu site rasterman.com. Epeg é um "thumbnailer" de geração automática mais rápido do planeta. Com uma API fácil de usar, pode ser integrado em qualquer aplicação que você desejar. O único inconveniente é que ele só manipula imagens JPEGs, mas não chega a ser um grande problema se levar em conta que todas as câmeras digitais disponíveis no mercado usam o formato JPEG como padrão.

Epsilon foi escrito por Atmos, inspirado pela velocidade da Epeg mas em resposta a uma necessidade de capacidade de "thumbnailing" de diversos formatos. Epsilon pode manipular JPEG, PNG, XCF, e GIF. Obviamente, já que ela não é uma biblioteca específica para manipular JPEG, não manipulará JPEG tão rápido quanto a Epeg, mas pode-se usar a própria Epeg para se ter as vantagens de velocidade que esta provê. Epsilon, diferente da Epeg, está em conformidade com o Thumbnail Managing Standard [http://triq.net/~jens/thumbnail-spec/index.html] do freedesktop.org. Portanto, ela direciona a saída dos thumbnails para a estrutura de diretório definida pela Thumbnail Managing Standard (~/.thumbnails/) ao invéz de um lugar definido pelo programador.

Ambas bibliotecas fazem tarefas tão expecíficas que as APIs são muito simples de usar. Epeg tem apenas 17 funções e Epsilon apenas 9, tornando-as fáceis de aprender para poder utiliza-las rapidamente.

# Receita: Thumbnailing simples com Epeg

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

A aplicação mais simplística de thumbnailing que podemos escrever usará apenas dois argumentos, o nome do arquivo ( imagem ) de entrada e o nome do arquivo ( thumbnail ) de saída. O seguinte exemplo de código usa Epeg exatamente para fazer isto.

### Exemplo 7.1. Um simples Thumbnail 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 é um exemplo bastante simplístico, não há checagem para certificar-se que o arquivo de entrada é realmente um JPEG, mas mostra adequadamente algumas das características da biblioteca. Ele pode ser compilado da senguinte maneira:

### Exemplo 7.2.

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

A função `epeg_file_open` abre um JPEG para ser manipulado, retornando um ponteiro para Epeg_Image. Este ponteiro pode então ser passado para as outras funções Epeg para manipulação.

Duas funções diferentes são usadas aqui para pegar algumas informações sobre a imagem de entrada: `epeg_size_get` e `epeg_comment_get`. Note que os valores retornados destas funções não são usadas em nenhuma outra função Epeg, são simplesmente para fins de informação. Um bom uso para estes valores retornados pode ser para definir o tamanho do thumbnail de saída, ou modificar e passar um comentário ao thumbnail de saída.

O conjunto de funções seguinte realmente fazem o trabalho. `epeg_decode_size_set` define o tamanho da saída do thumbnail. `epeg_file_output_set` define o nome do arquivo de saída. E `epeg_encode` faz o verdadeiro trabalho pesado. Observe que enquanto não checamos se houve exito aqui, `epeg_encode` retorna um int permitindonos checar se a operação foi um sucesso.

Uma vez criado o thumbnail, simplesmente chamamos a `epeg_close` para terminar o assunto.

Enquanto este exemplo é bem simples você pode ver como o básico funciona. Epeg tem também funções para redução, comentário no thumbnail, habilitar e desabilitar os comentários, conversão de espaço de cor e ajustes de qualidade que podem ser usados para conseguir os resultados que se deseja.

# Receta: Thumbnailing simples com Epsilon

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

Epsilon cria thumbnails em conformidade com a Thumbnail Managing Standard [http://triq.net/~jens/thumbnail-spec/index.html] da freedesktop.org. Thumbnails podem ser criados para uma variedade de formatos, incluindo suporte nativo para PNG, suporte para Epeg ou qualquer formato suportado pela Imlib2. Vejamos uma simples aplicação Epsilo, similar ao exemplo Epeg anterior.

### Exemplo 7.3. Um simples Thumbnail 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);
}
```

Pode ser compilado da seguinte maneira:

### Exemplo 7.4.

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

Notamos quase que imediatamente que não aceita-se nenhum nome de arquivo de saída, nem se usa nenhuma função de saída. A Thumbnail Managing Standard da reedesktop.org específica que todos os thumbnails são criados no diretório ~/.thumbnail. Este repositório cental de thumbnails permite compartilahr os thumbnails entre multiplas aplicações que aderem à específicação standard. Antes de compilar e executar o código exemplo, verifique se existe a imagem em ~/.thumbnails/large. Os thumbnails também se nomeiam de acordo com a específicação standard, renomeando o nome original com um MD5 checksum, de forma que o thumbnail não precisa ser refeito se a imagem está renomeada.

No nosso exemplo começamos verificando que obtemos uma imagem de entrada para fazer um thumbnail e então inicializamos o Epsilon usando a função `epsilon_init`. `epsilon_new` aceita um único argumento, a imagem que quer fazer um thumbnail, e retorna um ponteiro epsilon que é usado por outras funções.

Epsilon tem a habilidade de obter algumas informações básicas de suas imagens. No exemplo acima usamos `epsilon_info_get` para retornar uma estrutura Epsilon_Info contendo a data ( mtime ) de modificação da imagem de entrada, o lugar ( URI ), largura, altura e o tipo MIME. Aqui simplesmente damos a largura e altura da imagem usando os elementos `w` e `h` da estrutura info.

`epsilon_generate` é a função peso pesado. Esta função gerará o thumbnail e colocará no lugar apropriado. Seu valor de retorno indicando sucesso pode ser verificado usando as definições de macro CPP fornecida pelo header do Epsilon: `EPSILON_FAIL` e `EPSILON_OK`.

A limpeza é fornecida pela `epsilon_free`.

Epsilon, como visto aqui, é muito simples de usar e integrar com qualquer aplicação que faz uso de thumbnails. Não só fornece um API simples, mas integração com a reinante definição padrão para thumbnailing sem custo extra. Para

informação adicional sobre Epsilon, veja os documentos Doxygen do Epsilon em Enlightenment.org.

# Capítulo 8. Etox

Etox é uma biblioteca avançada de composição e layout de texto baseada no Evas, permitindo maiores e melhores funcionalidades que as disponíveis pelo Evas. É capaz de simplificar as tarefas de mostrar, mover, redimencionar, por em camadas e recortar texto, bem como alinhamento de texto, ajuste e modificação. Etox pode inteligentemente resolver obstáculos e aplicar *styles* predefinidos. Quase qualquer aspecto de nível textual pode ser controlado fácil e eficientemente com o Etox.

# Receta: Perspectiva geral de Etox

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

Para começar a usar Etox rapidamente é útil um exemplo simples. No seguinte código exemplo criamos uma Evas X11 usando Ecore_Evas e então colocaremos algum texto Etox nele.

### Exemplo 8.1. Exemplo 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 exmplo deve ser compilado da seguinte maneira:

## Exemplo 8.2.

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

A maior parte deste exemplo são funções standard do Ecore_Evas, assim nos concentraremos apenas nas partes relacionadas com Etox. Observe que usamos a função Evas `evas_font_path_append()` para definir o cominho de nossas fontes, isto é algo que o Etox não fará por você.

Seu texto Etox será sempre inicializado por adicionar um novo Etox usando a função `etox_new()` que devolve um Evas_Object. Já que o seu Etox é um objeto Evas, ele pode ser manipulado como tal. As funções de layout do Etox, como recortar e ajustar, são dependentes do tamanho do próprio Etox, portanto `evas_object_resize()` necessita ser chamada para definir o tamanho apropriado do Etox. Observer que a área do objeto *não* será igual por default ao tamanho do próprio Evas.

Etox usa o conceito de contextos. Um `context` é um conjunto de parâmetros como cor, fonte, alinhamento, estilo e marcas que são aplicadas a um certo conjunto de texto. Cada objeto Etox tem pelo menos um contexto associdado à ele que é criado quando se chama a função `etox_new()`. Por esta razão a função `etox_context_new()` *só* precisa ser chamada quando se deseja criar contexto adicionais.

Uma vez usado `etox_new()` para adicionar seu objeto Etox você precisa usar `etox_get_context()` para devolver um ponteiro à `Etox_Context` que pode então ser passado a outras funções de contexto para modificar os atributos do seu texto. No exemplo adicionamos a cor, fonte e o alinhamento de nosso contexto.

Das caracteríscas mais interessantes e simplistas do Etox são sua habilitade de inteligentemente quebrar o texto e de interpretar o caracter de nova linha do C ( \n ) como quebra de linha. Essas são duas características que o próprio Evas não proporciona, é da responsabilidade do programador certificar que o texto não saia do canvas.

A quebra inteligente de linha vem em duas formas que não são mutuamente exclusivas. A primeira é a quebra suave, que ajustará o texto quando um caracter exceder a largura do canvas. A segunda é a quebra de palavra, que irá ajustar o texto quando uma palavra exceder a largura do canvas. Tipicamente a segunda maneira é desaconselhada de maneira que obtenhamos "Isto é minha (quebra) string" no lugar de "Isto é mi(quebra)nha string". Note, ademais, que a quebra da palavra não funcionará a menos que a quebra suvae seja habilitada, portanto a quebra de palavra requer chamar *ambas* funções, `etox_set_soft_wrap()` e `etox_set_word_wrap()`.

Uma nota final sobre a quebra é que, por default tal ajuste irá inserir um `marcador de quebra de linha` na sua string de saída, um sinal "+" por default. Esta marca indica que um ajuste ocorreu e é impresso como primeiro caracter na linha seguinte. Sua string, portanto, se parecerá como: "Isto é minha (quebra) +string". Se preferir que o Etox quebre suavemente sem marcador, simplesmente configure o marcador como sendo vazio usando a função `etox_context_set_wrap_marker()`.

A string de texto Etox se configura usando `etox_set_text()`. É importante saber que a cadeia se aplica ao próprio Etox e não ao contexto. No há associação direta entre a string e o contexto que facilite a visualização do texto sem ter que modificar o contexto, ou vice-versa.

Enquanto este é um exemplo muito simples do uso de Etox, muito pode ser feito e como você pode ver a API é simples e limpa, preenchendo muitas necessidades de controle de texto que o Evas não possui.

# Capítulo 9. Edje

Edje é uma complexa biblioteca de desenho gráfico e layout. Seu propósito é abstrair todo elemento de interface da sua aplicação Evas do próprio código.

Uma aplicação Edje consiste em duas partes: O Código C que forma sua aplicação e uma Coleção de Dados Edje ( EDC ) que descreve cada elemento da sua interface. Ambas estão conectadas por sinais que são emitidas pelo EDC e recebidas por callbacks no código da sua aplicação. Usando este modelo de sinais o código fica completamente desinteressado no aspecto visual da sua interface, apenas recebe sinais. E como os sinais são processados por callbacks, no há necessidade que sua interface envie cada sinal disponível, fazendo possivel aplicações em grande escala e aplicações de tamanho "demo" com um único binário. Tanto faz se sua interface usa botões ou uma drag-bar para enviar dados, isto existe diferença para a sua aplicação.

# Receita: Um template para construir aplicações Edjes

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

O seguinte exemplo é um template que pode ser usado para iniciar rápida e fácilmente uma aplicação Edje. É semelhante ao template econtrado no capítulo sobre Evas, já que este também usa Ecore_Evas.

### Exemplo 9.1. Template 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 este template da seguinte maneira:

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

As chamadas importantes para vermos aqui estão localizadas no bloco Edje, seguindo a função `edje_init()`.

`edje_object_file_set()` define que EET Edje é usada assim como o nome da coleção à usar.

O resto das funções Edje/Evas no block Edje são necessárias para redimencionar a janela X11 para acomodar seu Edje. Começamos movendo a janela Evas e então pegando o tamanho mínimo do próprio Edje usando `edje_object_size_min_get()`. Então, usando `evas_object_resize()` podemos redimencionar o Edje, que é um objeto Evas real, para o tamanho do próprio evas. Adiante podemos mostrar o Edje e então redimencionar o próprio Evas (e graças ao Ecore a janela também) usando `ecore_evas_resize()`.

Depois disto callbacks podem ser adicionados para serem conectados à sua interface.

# Receita: Criando/Disparando callbacks Edje

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

As vezes é necessário avisar ao seu programa principal que algum evento aconteceu em sua interface de usuário. Mas você não quer partes da implementação misturando-se com o desenho da interface com o usuário. Isto é fácilmente feito com Edje disparando um sinal do seu programa EDC e associando um callback ao sinal no programa em C.

### Exemplo 9.2. Programa callback

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

int exit_cb(void *data, int type, void *ev);
void edje_cb(void *data, Evas_Object *obj,
                    const char *emission, const char *source);
```

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

{
    int ret = 0;
    Ecore_Evas *ee = NULL;
    Evas *evas = NULL;
    Evas_Object *edje = NULL;
    Evas_Coord w, h;

    if (!ecore_init()) {
        printf("error setting up ecore\n");
        goto EXIT;
    }
    ecore_app_args_set(argc, (const char **)argv);

    if (!ecore_evas_init()) {
        printf("error setting up ecore_evas\n");
        goto ECORE_SHUTDOWN;
    }

    if (!edje_init()) {
        printf("error setting up edje\n");
        goto ECORE_SHUTDOWN;
    }
    ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, exit_cb, NULL);

    ee = ecore_evas_software_x11_new(NULL, 0, 0, 0, 200, 300);
    ecore_evas_title_set(ee, "Edje CB example");
    ecore_evas_show(ee);

    evas = ecore_evas_get(ee);
    edje = edje_object_add(evas);
    edje_object_file_set(edje, "default.eet", "main");
    evas_object_move(edje, 0, 0);
    edje_object_size_min_get(edje, &w, &h);
    evas_object_resize(edje, w, h);
    ecore_evas_resize(ee, w, h);
    evas_object_show(edje);

    edje_object_signal_callback_add(edje, "foo", "bar", edje_cb, NULL);

    ecore_main_loop_begin();
    ret = 1;

    edje_shutdown();
ECORE_SHUTDOWN:
    ecore_shutdown();
EXIT:
    return ret;
}

int
exit_cb(void *data, int type, void *ev)
{
    ecore_main_loop_quit();
    return 1;
}

void
edje_cb(void *data, Evas_Object *obj,
                    const char *emission, const char *source)
{
    printf("got emission: %s from source: %s\n", emission, source);
}
```

A maior parte disto são inicializações padrões para o Ecore, Ecore_Evas e Edje. O callback é conectado com `edje_object_signal_callback_add(Evas_Object *o, char *emission, char *source, (void`

*)func(void *data, Evas_Object *obj, const char *emission, const char *source), void *user_data). O objeto o do qual o callback está conectado é um objecto Edje que foi criado no nosso arquivo EDC.

Os valores emission e source precisam ser strings que coincidam com a chamada emit no programa EDC que será visto mais tarde. Outra opção é por "*" em emission ou source. Isto fará com que o valor "*" coincida com qualquer coisa. Se deseja receber todas as chamadas que o edje emite, você pode colocar um "*" em ambos parâmetros, emission e source.

O func é a função à chamar e finalmente user_data é qualquer dado extra que você desejar ser passado para a callback.

O callback pode ser visto em edje_cb. Esta receberá os dados do usuário, o objeto edj que chamou o callback, e as strings emission e source.

Para ativar o callback nosso arquivo EDC necessita de um program que emitirá os requeridos emission e source.

## Exemplo 9.3. arquivo EDC

```
collections {
    group {
        name: "main";
        min: 200 100;

        parts {
            part {
                name: "bg";
                type: RECT;

                description {
                    rel1 {
                        relative: 0.0 0.0;
                        offset: 0 0;
                    }
                    rel2 {
                        relative: 1.0 1.0;
                        offset: -1 -1;
                    }
                    color: 255 255 255 255;
                }
            }
            part {
                name: "button";
                type: RECT;

                description {
                    rel1 {
                        relative: .4 .4;
                        offset: 0 0;
                    }
                    rel2 {
                        relative: .6 .6;
                        offset: 0 0;
                    }
                    color: 0 0 0 255;
                }
            }
        }
        programs {
            program {
                name: "down";
                signal: "mouse,down,*";
                source: "button";
                action: SIGNAL_EMIT "foo" "bar";
            }
```

```
        }
    }
}
```

A parte de interesse é `action: SIGNAL_EMIT "foo" "bar"`, isto fará o edje emitir uma emissão de `foo` com uma fonte de `bar`.

### Exemplo 9.4. Compilação

```
zero@oberon [edje_cb] -> edje_cc default.edc
zero@oberon [edje_cb] -> gcc -o cb main.c `ecore-config --cflags --libs` \
        `edje-config --cflags --libs`
```

Edje faz isto realmente simples para uma interface completamente abstraida da implementação. A única coisa que a interface precisa saber é enviar as emission e source corretas enquanto os eventos ocorrem.

# Receita: Trabalhando com arquivos Edje

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

Quando está trabalho com arquivos .edc e .eet você muitas vezes precisa transformar um arquivo no outro. Em ajuda, Edje fornece um grupo de ferramentas para facilitar estas transformações.

Os programas disponíves são:

| | |
|---|---|
| edje_cc | Compila um arquivo EDC, imagens e fontes, dentro de um arquivo EET |
| edje_decc | Descompila um arquivo EET retornando o arquivo EDC, as imagens e as fontes |
| edje_recc | Recompila um arquivo EET |
| edje_ls | Lista os grupos em um arquivo EET |
| edje | Mostar os grupos em um arquivo EET |

Cada um destes programas são discutidos com mais detalhes abaixo:

## edje_cc

`edje_cc` é um dos principais programas Edje que você irá usar. Ele é responsável pela compilação dos seus arquivos EDC, incluindo imagens e fonts, dentro dos arquivos EET correspondentes.

### Exemplo 9.5. Uso do edje_cc

```
edje_cc [OPÇÕES] arquivo_de_entrada.edc [arquivo_de_saida.eet]
```

### Opções

| | |
|---|---|
| -id *image/directory* | Adiciona um diretório como path relativo para procurar pelas imagens |
| -fd *font/directory* | Adiciona um diretório como path relativo para procurar pelas fontes |
| -v | Saída em modo prolixo (detalhado) |
| -no-lossy | Não permitir imagens com lossy compression 1 |
| -no-comp | Não permitir que as imagens sejam armazenadas com lossless compression 2 |
| -no-raw | Não permitir que as imagens sejam armazenadas com zero compressão (imagem crua) |
| -min-quality *VALOR* | Não permitir imagens lossy1 com qualidade < VAL (0-100) |
| -max-quality *VAL* | Não permitir imagens lossy1 com qualidade > VAL (0-100) |
| -scale-lossy *VAL* | Escalar image lossy1 por este fator de porcentagem(0 - 100) |
| -scale-comp *VAL* | Escalar imagem lossless2 por este fator de porcentagem(0 - 100) |
| -scale-raw *VAL* | Escalar imagem sem compressão (imagem crua) por este fator de procentagem(0 - 100) |
| -D*define_val=to* | Definições ao estilo CPP, para definir entrada de macros para o fonte .edc |

# edje_decc

`edje_decc` permite descompilar arquivo EET de volta para EDC bem como as imagens e fontes. Isto facilita distribuir seu fonte para fazer o arquivo EET sempre que necessitar e o usuário final terá acesso ao código e ao produto final.

### Exemplo 9.6. Uso do edje_decc

```
edje_decc arquivo_de_entrada.eet
```

# edje_recc

`edje_recc` permite recompilar um arquivo EET sem necessariamente descompilar primeiro. Isto lhe permite modificar os parâmetros passado pelo `edje_cc` para modificar o visual e as exigências do tamanho.

### Exemplo 9.7. Uso do edje_recc

```
edje_recc [OPÇÕES] arquivo_de_entrada.eet
```

### Opções

| | |
|---|---|
| -v | Saída em modo prolixo (detalhado) |

1lossy compression é a técnica de compressão que parte dos dados são perdidos, esta técnica tenta elimiar informações redundantes ou desnecessárias causando perda de qualidade

2lossless compression a técnica de compressão que permite que os dados sejam descomprimidos sem nenhuma perda de informações, ou seja os dados reconstruídos são exatamente iguais aos dados originais.

| | |
|---|---|
| -no-lossy | Não permitir imagens com lossy compression1 |
| -no-comp | Não permitir que as imagens sejam armazenadas com lossless compression2 |
| -no-raw | Não permitir que as imagens sejam armazenadas com zero compressão (imagem crua) |
| -min-quality *VAL* | Não permitir imagens lossy1 com qualidade < VAL (0-100) |
| -max-quality *VAL* | Não permitir imagens lossy1 com qualidade > VAL (0-100) |

# edje_ls

`edje_ls` fornece uma listagem de todos os grupos dentro de um arquivo EET informado. Este é um jeito rápido de ver o que há disponível no EET informado.

### Exemplo 9.8. Uso do edje_ls

```
edje_ls [OPÇÕES] arrquivo_de_entrada.eet ...
```

### Opções

| | |
|---|---|
| -o *arquivodesaida.txt* | Direciona a listagem das coleções para um arquivo |

# edje

`edje` também é um dos principais programas que você irá usar. `edje` permite você ver cada um dos grupos do seu programa. Ele permite você ver como os items estão sendo vistos e como eles reagem aos sinais.

### Exemplo 9.9. Uso do edje

```
edje file_to_show.eet [OPÇÕES] [que_coleções_mostrar] ...
```

### Opções

| | |
|---|---|
| -gl | Usa OpenGL para renderizar |
| -g *W*x*H* | Ajuste a geometria da janela para WxH (Largura x Altura) |
| -fill | Faz o item ocupar toda a janela |

Estas cinco ferramentas simples devem auxiliar na construção e manutenção de seus arquivos Edje e EETs. Eles também facilitam a recuperação do fonte que compreende um arquivo EET informado, tornando fácil o aprendizado de como desempenham trabalhos diferentes.

# Capítulo 10. Edje EDC e Embryo

Os arquivos fontes de Coleções de Dados Edje ou Edje Data Collections ( EDC ) permitem uma fácil criação de ricas e potentes interface gráfica. Suas aplicações Edje está dividida em duas partes distintas, o código da aplicação ( usando chamdas de `Edje.h` ) e a descrição da interface em EDC. A única conectividade requerida entre sua interface e o código da sua aplicação são os sinais emitidos por sua interface que são recebidas por callbacks Edje no código da sua aplicação.

Um EDC está dividio em várias seções maiores descrevendo as imagens e fontes que são usadas na interface, descrições de como as várias partes ( `part` ) da interface são dispostas, e descrições de ações ou `program` que ocorrem quando se interage com sua interface. Esta funcionalidade pode ser suplementada usando a linguagem de script Embryo para adicionar programabilidade no estilo C na própria EDC Edje.

O resultado final de uma EDC, incluindo todas as fonts e imagens, é um único EET. Já que a interface completa é disponível em um único arquivo de "tema" é drásticamente simplificada.

Embora os EDC Edje possam ser vistos como "temas" eles são muito mais que isto. Um "tema" tradicional é um arquivo ou grupo de arquivos que incrementam uma interface gráfica existente mudando a cor dos elementos ou modificando as imagens que compoem a própria interface. Mas estes métodos são insuficiente para realmente mudar o design da interface de uma aplicação, limitando os criadores de temas de modifica-los e impossibilitando um redesenho da aplicação em algum ponto para expandir as capacidades da interface para uma maior funcionalidade. Uma aplicação GTK sempre terá um aspecto semelhante independente do tema que usar. Um exemplo simples é que uma aplicação GTK ou QT sempre terá uma forma retangular e se tem uma borda não se pode tirá-la mudando o tema. No entanto, uma aplicação Edje poderá mudar da forma retangular para oval com uma simples modificação da EDC, ou poderá remover e rearrajnar todos os elementos da interface sem nunca tocar no código. Desta maneira Edje permite uma grande quantidade controle e uma flexibilidade maior que qualquer outra solução na comunidade Open Source e permite um modelo Aberto de programação para permitir que não programadores ( como muitos criadores de temas são ) contribuam e modifiquem as coisas como desejarem.

# Receita: Comutador Edje/Embryo

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

No início Raster [http://www.rasterman.com] fez o Edje, e isto era bom. Os homens das cavernas que descobriram o Edje nas paredes das cavernas (#edevelop) ficaram maravilhados, mas logo perceberam muitos inconvenientes. Tendo uma quantidade certa de criatividade você poderia fazer coisas, comutadores por exemplo, mas precisava-se usar alquimia para fazer corretamente. Para fins históricos, é fornecido um comutador Edje sem Embryo. Veja Edje sem Embryo mais abaixo.

Você observará que precisa-se falar em sinais com a sua aplicação para determinar o estado do seu comutador. Então, sem mais demora, aqui vai um comutador Edje usando Embryo, de uma maneira *muito* mais elegante.

O script Embryo dentro do Ejde, no doravante script EE, te dá variáveis. Você pode ter inteiros, números de ponto flutuante e strings. Isto significa basicamente que pode ter alguma lógica de programação nos teus edjes. Nada complexo como estruturas mas variáveis simples contidas em um grupo poderiam assemelhar-se aos membros de estruturas.

A primeira parte do EE é escolher as suas variáveis. Neste simples exemplo só temos uma variável, e você envolveu-a em um group edje declarando um bloco *script { ... }*. *button_toggle_state* é implicitamente um inteiro, e será usando como vaiável boolena para permitirnos saber se o botão de comutação está ativado ou desativado. A parte legal desta variável é que podemos usá-la como uma forma de comunicação entre nossa aplicação e nosso edje. Ademais você pode tranquilizar-se sabendo (assumindo que você fez isto corretamente) que nenhuma artimanha do ejde lançará sua aplicação ao limbo.

### Exemplo 10.1. Criando a variável

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

A segunda aprte do escript EE é inicializar as variáveis. Na maioria das vezes pode-se assumir que estas variáveis serão inicializadas com zero, mas é um bom costume você mesmo inicializa-las. O Edje emite um sinal "load" quando o grupo é carregado na memória, esta é a sua oportunidade de iniciar as variáveis embryo.

### Exemplo 10.2. Inicializando variáveis

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

A terceira parte é propriamente dar um aspecto ao seu edje. Para este exemplo é usado retângulos, mas imagens e textos também funcionam corretamente. Há um objecto de background para consistência, e há um retângulo chamado "toggler" (comutador). toggler tem dois estados, o estado default ( implicitamente desabilitado ) e um habilitado. Quando toggler é clicado deverá mudar para o outro estado. off -> on, on -> off. toggler terá uma cor vermelha quando desabilitado e quando habilitado será azul, de modo que possa ser facilmente diferênciado. O background será branco porque não pode ser vermelho e nem azul :D

### Exemplo 10.3. O botão toggler

```
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);
                }
            }
        }
    }
}
```

A quarta parta está conectando-se aos eventos de mouse para disparar a comutação. Não apenas mudando a variável Embryo, mas também mudando a aparência do nosso ejde. Este exemplo usa o click normal do botão esquerdo do mouse para mudar o estado do comutador, nos termos do edje "mouse,clicked,1". Este exemplo não usa a chamada a função incorporada no Embryo *set_state*, mas emite sinais que são recebidas por outros programas. A razão por trás disto é nos permitir transições visuais suaves entre os dois estados. A chamada à função Embryo set_state é uma mudança de estado imediato, e não tem um aspecto tão agradável como a transição SINUSOIDAL usada nos seguintes framentos.

### Exemplo 10.4. Conectando-se com os eventos de mouse

```
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;
             }
         }
    }
}
```

A quinta parte esta ponderando o senário apresentado. Isto é apenas a ponta da iceberg no que diz respeito a scripts EE. Você pode adicionar mais variávels para elevar o estado interno que não está completamente relacionado com sua aplicação. Há nuances entre isto e o uso prático das variávels Embryo, ademais, entendendo estes blocos de contrução será mais simples escrever ou trabalhar com aplicações scripts EE.

- O que há de errado com a técnica aqui apresentada?
- O que acontece se a aplicação quer o comutador "on" por default?

Você pode usar um script similar abaixo para construir este exemplo.

### Exemplo 10.5. Contruir o 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
```

### Exemplo 10.6. Comutador Edje sem 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;
            }
        }
    }
}
```

# Receita: Efeito de dilução Edje no texto

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

Os efeitos de texto podem dar uma aparência legal ao seu programa. Mas e se caso você desejar incorporar nestes efeitos um efetio de diluição? Edje faz isto possível e relativalmente simples.

Todo o que você precisa fazer é incorporar o atributo `color3` conforme vai diluindo o atributo `color` do texto. `color3` mudará os valores de cor do efeito.

Isto é ilustrado no seguinte exemplo:

## Exemplo 10.7. Diluindo efeito com texto

```
collections {
    group {
        name, "Main";
        min, 30 30;

        parts {
            part {
                name, "foo";
                type, TEXT;
                effect, SOFT_SHADOW;
                mouse_events, 1;

                description {
                    state, "default" 0.0;
                    rel1 {
                        relative, 0 0;
                        offset, 0 0;
                    }
                    rel2 {
                        relative, 1.0 1.0;
                        offset, -1 -1;
                    }

                    text {
                        text, "foo text";
                        font, "Vera";
                        size, 22;
                    }
                    color, 255 255 255 255;
                    color3, 0 0 0 255;
                }
                description {
                    state, "out" 0.0;
                    rel1 {
                        relative, 0 0;
                        offset, 0 0;
                    }
                    rel2 {
                        relative, 1.0 1.0;
                        offset, -1 -1;
                    }

                    text {
                        text, "foo text";
                        font, "Vera";
                        size, 22;
                    }
                    color, 0 0 0 0;
                    color3, 255 255 255 0;
                }
            }
        }
        programs {
            program {
                name, "foo";
                signal, "mouse,in";
```

```
                source, "foo";
                action, STATE_SET "out" 0.0;
                transition, SINUSOIDAL 2.0;
                target, "foo";
            }
            program {
                name, "foo";
                signal, "mouse,out";
                source, "foo";
                action, STATE_SET "default" 0.0;
                transition, SINUSOIDAL 2.0;
                target, "foo";
            }
        }
    }
}
```

Este exemplo pode ser compilado para dentro de um `.eet` com o seguinte comando.

### Exemplo 10.8. Compliação

```
zero@oberon[edje_text] -> edje_cc text.edc
```

Por alterar o valor de `color3` com o valor de `color` você será capaz de alterar a aparância dos seus efeitos com seu texto.

# Capítulo 11. EWL

A Enlightened Widget Library (EWL) é um conjunto de ferramentas (toolkit) de widget desenvolvidas sobre as outras outras bibliotecas da EFL. Ewl usa Evas como backend de renderização e sua aparência é abstraída para ser controlada pelo Edje.

EWL é similar a muitos outros toolkits entre eles o GTK, QT e MOTIF. As APIs se diferem, mas os conceitos são os mesmos.

# Receta: Introducción a EWL

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

Mediante o uso da Enlightened Widget Library (EWL), uma porção maior de potência pode ser posta nas mãos dos programadores com pouco esforço.

Esta introdução á EWL mostrará como criar uma simples aplicação para visualização de texto com uma barra de menu e um dialogo de arquivo. A área de texto terá barras de rolagem e permitirá também rolar usando as teclas do teclado ou a roda do mouse.

### Exemplo 11.1. Includes e declarações

```
#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);
```

O include necessário para escrever uma aplicação Ewl é <Ewl.h>. Declaramos a janela principal e o dialogo de arquivo como globais para facilitar o acesso nas funções de callback. Elas não precisam ser globais, mais para o propósito desta exemplo é mais simples que sejam.

### Exemplo 11.2. main

```
/* lets go */
int main(int argc, char ** argv) {
    ewl_init(&argc, argv);
```

```
    mk_gui();
    ewl_main();
    return 0;
}
```

A função principal para nosso visualizador de texto é muito simples. Começamos inicializando o ewl mediante a chamada `ewl_init()`. Ewl pega argc e argv para ler alguns parâmetros de linha de comando. Isto inclue coisas como iniciar o tema que vai usar ( --ewl-theme ) ou iniciar o motor de renderização ( --ewl-software-x11, --ewl-gl-x11, etc. ).

`ewl_init()` se encarrega de todo o trabalho de inicializar as outras bibliotecas requeridas, abstraindo isto do programador em uma interface simples.

A chamada a mk_gui inicializará a janela principal e qualquer conteúdo requerido.

A chamada a `ewl_main()` inicia o laço principal de processo, e ao terminar controla toda a finalização requida pela aplicação, assim sendo, não há chamada de finalização na nossa rotina principal.

## Exemplo 11.3. mk_gui: Criar a janela

```
/* 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);
```

A primeira coisa que precisamos fazer para executar nossa aplicação é criar a janela principal da aplicação. Isto é feito por meio da chamada a `ewl_window_new()`. Uma vez com a janela continuamos informando o título (como aparecerá na barra de título da janela do Gerenciador de Janelas), o nome e a classe da janela.

Uma vez informada as informações principais da janela, solicitamos o tamanho default da janela de 200x300 por meio da chamada `ewl_object_size_request`. Junto com o tamanho default, podemos informar o tamanho máximo e mínimo da janela atravéz da chamada `ewl_object_minimum_size_set` e `ewl_object_maximum_size_set`. Mas como isto não é necessário para esta aplicação eles ficarão de fora.

A configuração final da janela é feita selecionando a politica de preenchimento com `ewl_object_fill_policy_set`. Isto ajusta a forma como o Ewl empacotará os widgets na janela, com um dos possíveis valores:

| | |
|---|---|
| EWL_FLAG_FILL_NONE | Não preencher ou encolher em nenhuma direção |
| EWL_FLAG_FILL_HSHRINK | Encolher horizontalmente |
| EWL_FLAG_FILL_VSHRINK | Encolher verticalmente |

| | |
|---|---|
| EWL_FLAG_FILL_SHRINK | Encolher tanto horizontal como verticalmente |
| EWL_FLAG_FILL_HFILL | Preencher horizontalmente |
| EWL_FLAG_FILL_VFILL | Preencher verticalmente |
| EWL_FLAG_FILL_FILL | Preencher tanto horizontal como verticalmente |
| EWL_FLAG_FILL_ALL | Encolher e Preencher de uma vez |

Após definir todas as propriedades da janela anexa-se um callback para capturar a destruição da janela principal com `ewl_callback_append()`. A função `destroy_cb()` será chamada caso alguém requeira que a janela seja destruida de alguma maneira.

Mostramos a janela principal com uma chamada a `ewl_widget_show()`. Se `ewl_widget_show()` não for chamada nada aparecerá na tela. Todos os widgets estão invisíveis até que explicatamente solicitemos mostra-los. Em oposição a isto é a `ewl_widget_hide()` que removerá um widget da tela.

## Exemplo 11.4. O container 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);
```

Podemos empacotar todos nossos widgets na própria janela principal, mas isto pode causar problemas se desejarmos mudar as coisas facilmente, então criaremos uma caixa dentro da janela principal para que mantenha todos os nossos widgets.

Isto é feito criando uma caixa vertical com `ewl_vbox_new()`. Então se toma a caixa e acrescenta a lista de filhos da janela principal com `ewl_container_child_append()`. Depois de anexar a janela selecionamos uma politica de preenchimento para preencher tanto na horizontal como na vertical com `ewl_object_fill_policy_set`, e mostramos o widget com `ewl_widget_show()`.

A ordem em que vai se pondo os widgets no container afetará a maneira como se mostra a aplicação. O primeiro widget empacotado será o primeiro widget a ser mostrado. Como especificamos uma caixa vertical empacotaremos os widgets começando do topo até o fundo da nossa tela.

## Exemplo 11.5. Criar a barra de menu

```
/* 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_spacing_set(EWL_BOX(menu_bar), 4);
ewl_object_padding_set(EWL_OBJECT(menu_bar), 5, 5, 5, 5);
ewl_widget_show(menu_bar);
```

O primeiro widget que adicionamos é a barra de menu. Colocaremos o próprio conteudo do menu mais tarde, depois que os outros widgets tenham sido criados, mas precisamos adicionar a barra de menu primeiro.

As chamadas são as mesmas que vismos acima, adicionar container pai, iniciar a politica de preenchimento, mostrar o widget. As que não foram vitas antes incluem `ewl_object_alignment_set()`, que informa como o widget será alinhado dentro do container. Neste caso estamos usando EWL_FLAG_ALIGN_LEFT, mas podemos usar qualquer outro dos alinhamentos incluindo:

- EWL_FLAG_ALIGN_CENTER
- EWL_FLAG_ALIGN_LEFT
- EWL_FLAG_ALIGN_RIGHT
- EWL_FLAG_ALIGN_TOP
- EWL_FLAG_ALIGN_BOTTOM

Assim o menu irá alinhar com o lado esquerdo da caixa principal.

Então especificamos o espaçamento dos itens do menu. Isto dará um pouco mais de espaço entre os nossos itens de menu, isto é feito com `ewl_box_spacing_set()`. Depois de mudar o espaço mudamos o preenchimento ao redor da caixa como um todo com uma chamada à `ewl_object_padding_set()`, isto incrementará uma quantidade de espaço em volta do widget.

### Exemplo 11.6. Criar o painel de rolagem

```
/* 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);
```

O painel de rolagem vai ser o pai do nosso objeto texto. O painel de rolagem nos dá toda a mágica para controlar as barras de rolagem e a sua própria rolagem.

O painel de rolagem é criado com uma chamada à `ewl_scrollpane_new()`, e então anexamos o painel de rolagem na caixa principal e selecionamos a sua política de preenchimento.

As chamadas à ewl_scrollpane_[hv]scrollbar_flag_set() dizem ao Ewl como deverão compartar-se as barras de rolagem. Os valores possíveis são:

- EWL_SCROLLBAR_FLAG_NONE
- EWL_SCROLLBAR_FLAG_AUTO_VISIBLE
- EWL_SCROLLBAR_FLAG_ALWAYS_HIDDEN

Uma vez que as barras de rolagem tenham sido iniciadas pedimos ao Ewl que mostre o widget.

### Exemplo 11.7. Criar a área 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);
```

A área de texto será responsável em manter o texto no nosso visualizador. O widget é criado com uma simples chamada à `ewl_text_new()`. Isto causará a criação da área de texto, mas com um texto em branco. Como na barra de menu, incrementamos o preenchimento ao redor da área de texto para dar um pouco de espaço entre o texto e os outros elementos.

## Exemplo 11.8. Criar o conteúdo do menu

```
    /* 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);
    }
```

Agora que a área de texto foi criada podemos continuar por criar as entradas do menu. Fiz isto dentro do seu próprio bloco para limitar o número de declarações no início da função, isto não é requerido por nenhuma razão.

O menu é criado com uma chamada à `ewl_imenu_new()`. Esta função pega dois parâmetros, o primeiro é a imagen para mostrar com este menu, neste casso NULL, sou seja, sem imagem. O segundo parâmetro é o nome do menu tal como aparecerá na barra de menu.

Uma vez criado o menu, pode continuar por adicionar os itens do menu por meio da uma chamada à `ewl_menu_item_new()`. Esta novamente pega dois parâmetros, o ícone para mostrar junto com esta entrada no menu, e o nome que aparecerá no menu.

Ao passo que os elementos são adicionados no menu fazemos uma chamada à `ewl_callback_append()` para conectar à chamada EWL_CALLBACK_SELECT. A função dada será executada quando o usuário clicar no item de menu. No caso de "open" tivemos que passar a área de texto ao callback de open para permitirmos modificar facilmente seu conteúdo.

Outros menus poderão ser adicionados da mesma maneira, mas para esta aplicação apenas um menu é necessário.

## Exemplo 11.9. Vincular callbacks

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

Uma vez que tudo está iniciado na janela principal, adicionamos os callbacks que desejamos receber. Neste caso es-

tamos vinculando-nos ao callback EWL_CALLBACK_KEY_UP. Não necessitamos fazer nada para que a roda do mouse deslize o painel de rolagem pois isto é configurado no próprio painel de rolagem.

### Exemplo 11.10. callback de finalização

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

Quando a janela principal é fechada, destruímos o widget que está nela mediante uma chamada à `ewl_widget_destroy()`. Depois que a janela é destruída solicitamos ao Ewl que desejamos saír chamando a `ewl_main_quit()`. Isto fará com que o Ewl pare o loop de processamento principal e retornará da chamada prévia à `ewl_main()`.

### Exemplo 11.11. Callback do item open do menu 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);
}
```

Se um usuário clicar no ítem "Open" do menu "File" a função `file_menu_open_cb()` será executada. Quando isto ocorrer necessitamos criar o diálogo de arquivo para que o usuário selecione o arquivo para ver.

Da mesma forma que na janela principal, criamos uma janela para conter o diálogo de arquivo e informamos um título, tamanho e classe. Solicitamos um tamanho default, selecionamos sua política de preenchimento e adicionamos um callback para controlar a destruição da própria janela. Então adicionamos uma caixa simples dentro da janela para conter o diálogo de aquivo.

Uma vez configurada a janela, fazemos uma chamada para criar o diálogo de arquivo. Isto é feito com uma chamda à `ewl_filedialog_new()`, especificando o tipo de diálogo de arquivo que queremos cirar. Neste caso queremos um diálogo que nos possíbilite abrir um diretório, então especificamos EWL_FILEDIALOG_TYPE_OPEN. Poderemos especificar EWL_FILEDIALOG_TYPE_SAVE se desejarmos usar o diálogo para salvar o arquivo ao invéz de abri-lo.

Então continuamos criando um botão extra para permitir o usuário que navegue no seu diretório pessoal com um simples click. Isto é feito chamando `ewl_button_new()` e condicionando o botão no próprio diálogo de arquivo.

### Exemplo 11.12. Callback de finalização do diálogo de arquivo

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

Quando precisamos nos livrar do diálogo de arquivo, eliminamos o widget da tela com uma chamada à `ewl_widget_hide()`, e uma vez que não é mais mostrado destruímos o widget com uma chamada à `ewl_widget_destroy()`.

### Exemplo 11.13. Callback do botão open do diálogo de arquivo

```
/* 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 executando quando o usuário clicar no botão "Open" no diálogo de arquivo, ou quando o usuário der um duplo clique em um arquivo no diretório. O evento passado (o parâmetro ev) será o caminho completo do arquivo que o usuário selecionou.

No nosso caso, tomamos esse arquivo e o passamos à função para ler o arquivo e devolver o texto do arquivo. Então usando este texto como está, chamamos a função `ewl_text_text_set()` que colocará o texto no objeto de texto dado.

Já que o usuário terminou sua seleção, o diálogo de arquivo é escondido.

### Exemplo 11.14. Callback do botão home do diálogo de aquivo

```
/* 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);
}
```

Se o usuário clicar no botão "Home" no diálogo de aquivo queremos mostrar o conteúdo do seu diretório pessoal. Colocamos o diálogo de arquivo como dado do callback, assim o pegamos no callback e capturamos o diretório pessoal da variável de ambiente. A chamada para `ewl_filedialog_set_directory()` muda o diretório atual que está mostrando no diálogo de arquivo para ser o diretório pessoal do usuário.

### Exemplo 11.15. Ler o arquivo 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 é uma rotina simples para tomar o arquivo dado, abrir-lo e ler seu conteúdo na memoria. Provavelmente não é a melhor ideia para uma aplicação real, mas é o suficiente para este programa exemplo.

### Exemplo 11.16. Callback 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 chamda quando o usuário solta uma tecla do teclado. O callback receberá uma estrutura de evento Ewl_Event_Key_Down contendo a informação da própria tecla pressionada. No nosso caso só necessitamos da entrada keyname que é o nome da tecla pressionada.

Se o usuário pressinar a tecla "q" simplesmente chama o callback destroy e terminamos com isto.

"Left", "Right", "Up" e "Down" são as teclas de cursor do teclado. Se alguma destas teclas forem pressionadas froçamos o painel de rolagem a rolar na direção especificada.

Para controlar o painel de rolagem necessitamos saber onde ele está atualmente no arquivo e a distância que deverá rolar cada incremento/decremento. Por sorte Ewl torna isto fácil. A chamada à ewl_scrollpane_[hv]scrollbar_value_get() devolverá o valor atual da barra de rolagem. Este é um valor do tipo double com limites [0, 1] inclusivo. Um valor de 0 significa que a barra de rolagem está no topo e um valor de 1 que está no fundo. Esquerda e direita funcionam da mesma forma, com 0 sendo esquerda absoluta e 1 direita absoluta.

A segunda parte da informação é obtida mediante a chamda à ewl_scrollpane_[hv]scrollbar_step_get(). O passo (step) é a distância que rolará o painel de rolamento com uma interação. Usando estes dois valores podemos então mover a barra de rolagem da direção correta com a chamada à ewl_scrollpane_[hv]scrollbar_value_set().

## Exemplo 11.17. Compilação

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

Compilar uma aplicação Ewl é tão simples como chamar ewl-config e obter os --cflags e --libs.

Então é isto. Com este exemplo você deverá ter uma aplicação Ewl funcionando plenamente, incluindo menus, um diálogo de arquivo e uma área de texto com barras de rolagem horizontais e verticais. Este exemplo apenas arranha a superfície da pontência contida dentro do conjunto de ferramentes Ewl com muitos outros tipo de widget disponiveis para usar.

# Capítulo 12. Evoak

Evoak é um servidor de canvas. Similiar a um servidor X que serve um display e operações gráficas. Evoak server um único canvas para ser compartilhado por várias aplicações ( clientes ) permitindo que cada uma delas manipule seu próprio conjunto de objetos no canvas.

# Receita: Cliente Evoak hello

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

Esta receita é uma introdução muito simples ao mundo da programação Evoak. Seguindo a grande tradição, esta receita mostrará a versão canadense de "Hello World" num canvas evoak.

### Exemplo 12.1. Includes e pré-declarações

```
#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 precisamos incluir o header Evoak, e o header Ecore é necessário para acessar as funções de callback.

### Exemplo 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 precisa ser iniciado com uma chamada à `evoak_init`. Isto iniciará todas as bibliotecas internas e os requerimentos para a Evoak.

Assim que o Evoak iniciar corretamente, conectaremos dois callbacks, o primeiro é para informação de canvas e o

segundo é caso formos desconectados do servidor Evoak. Isto será discutido depois quando os próprios callbacks forem mostrados.

Uma vez que os callbacks estão em seus lugares precisamos conectar ao servidor de canvas Evoak. Esto é feito mediante a chamada para `evoak_connect`. Os parâmetros para `evoak_connect` são: o servidor para conectar, o nome do cliente e a classe do cliente. Se o primeiro argumento for NULL, como no exemplo, o servidor Evoak default é coenctado. O segundo arqumento da `evoak_connect` é o nome do cliente, este valor deverá ser único já que isto será usado para distinguir um cliente do outro. O argumento final, a classe, é o tipo do cliente, alguns dos possíveis valores são: "background", "panel", "application" ou "custom".

Se a chamada para `evoak_connect` falhar um valor NULL será retornado. Assim, sempre que recebemos um objeto Evoak, iniciamos o loop principal ecore. Quando ecore termina, chamamos `evoak_disconnect` para disconectarmos do servidor Evoak.

Terminamos por chamar a função `evoak_shutdown` para a limpeza final.

### Exemplo 12.3. Callback de informação 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;
}
```

Um callback de informação de canvas será chamado quando nosso cliente recebe informação do servidor de canvas Evoak. Com esta informação podemos fazer a inicialização do conteúdo dos nossos clientes. Isto está contido dentro da flag setup_called já que só queremos inicializar uma vez.

### Exemplo 12.4. Callback de desconexão

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

A callback de desconexão será chamado quando o cliente for desconectado do servidor Evoak. Neste caso, uma simples solução de sair é usado.

### Exemplo 12.5. rotina 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);
}
```

A rotina de setup será chamada uma vez para inciar a janela de nosso cliente. Para este exemplo, o cliente apenas desenha o texto 'Hello Evoak, eh'.

A primeira coisa que fazemos é chamar `evoak_freeze`, isto deve nos proteger de ficar recebendo qualquer callback não solicitado enquanto iniciamos nossa interface. Ao final da função chamamos a recíproca `evoak_thaw` para "descongelar".

Então continuamos criando um objeto de texto com `evoak_object_text_add` e com este objeto de texto, iniciamos a fonte, cor e conteúdo do texto com chamadas para `evoak_object_text_font_set`, `evoak_object_color_set` e `evoak_object_text_text_set` respectivamente.

## Exemplo 12.6. Compilação

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

Como com tantas outras bibliotecas baseada en EFL, compilar uma aplicação Evoak é tão simples como chamar o programa evoak_config e obter os conteúdos de --cflags e --libs.

Isto é tudo, foi uma introdução à Evoak realmente simples e a superfície permanece sem nenhum arranhão referente ao potêncial disponível para aplicações cliente.

# Capítulo 13. Emotion

Emotion é uma biblioteca de objetos de vídeo e mídia desenhada para interagir com Evas e Ecore fornecendo de forma autonoma objetos de "video" e "audio" que podem ser movidos, redimensionados e posicionados como qualquer outro objeto normal, mas podem reproduzir video e audio e podem ser controlados atravéz uma API de controle de alto nível permitindo ao programdor montar rápidamente um sistema multimídia com um mínimo trabalho. Emotion fornece um sistema de camadas de decodificadores modular onde um módulo decodificador pode ser plugado separadamente para prover recursos de decodificação. Emotion atualmente tem 1 módulo decodificardor que usa XINE como decodificador, permitindo-o reproduzir DVD's, MPEG's, AVI's, MOV's, WMV's e muito mais. Este programa teste é um útil reprodutor de DVD ( sem uma interface bonita ) que pode reproduzir várias fontes de vídeo com semi-translucência entre outras coisas.

# Receita: Player DVD com Emotion

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

Para mostrar como o Emotion fácilmente pôe um arquivo de vídeo, DVD, VCD ou outro conteúdo de midia em um canvas dê uma olhada no seguinte programa. É um reprodutor de DVD completo, mas muito simples. Possui um controle por mouse limitado, não controla as mudanças relacionadas ao aspcto e etc. Tudo em 55 linhas de código C.

Todo o código nesta e na próxima receita pode ser compilado usando:

### Exemplo 13.1. Compilação

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

### Exemplo 13.2. Reprodutor de DVD em 55 linhas 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();
}
```

Agora temos uma introdução muito simples ao Emotion. Este fragmento de código pode ser facilmente expandido para trabalhar com qualquer formato de midia suportado pelo Emotion, bem como tratar relações do aspecto, navegação pelo teclado, e mais.

# Receita: Player de midia expandido com Emotion

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

Expandindo nossa receita anterior, podemos fazer o emotion redimencionar apropriadamente (mantendo a relação de aspecto).

### Exemplo 13.3. Reprodutor de midia 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();
}
```