Le Livre de Recettes EFL

par Various et Ben 'technikolor' Rockwood

Le Livre de Recettes EFL

par Various et Ben 'technikolor' Rockwood

Stuff.

Table des matières

Liste des exemples

Chapitre 1. Introduction

Bienvenue dans le monde lumineux de la programmation. Ce livre de cuisine est une collection de trucs et astuces, de tutoriaux et d'introductions élaboré sous forme de recettes dans le but de vous aider à devenir rapidement compétent avec les bibliothèques de base d'Enlightenment. Les "Enlightenment Foundation Libraries", appelées plus communement EFL, sont un ensemble de bibliothèques originellement écrites afin de servir au window manager Enlightenment DR17. Cependant, au cours du développement de ces bibliothèques, des fonctionnalités plus généralistes ont été ajouttées, donnant lieu à un ensemble riche et puissant capable de résoudre toutes sortes de problèmes et de constituer une véritable alternative aux populaires GTK et QT.

EFL est un groupe exhaustif de bibliothèques C qui penvent subvenir à vos besoins graphiques sur presque n'importe quelle plateforme. Ce qui suit est une décomposition concise des bibliothèques qui composent EFL.

Liste des composants EFL

Imlib2	Bibliothèque complète de manipulation d'image, incluant le rendu de surface d'affichage X11.
EVAS	Bibliothèque de canvas avec comprenant plusieurs moteurs dont l'accelération matérielle OpenGL.
Ecore	Bibliothèque modulaire permettant de manipuler les événements et les temporisateurs, mais gérant aussi les sockets, IPC, FB et X11, la gestion des jobs, de la configuration et plus encore.
EDB	Une bibliothèque de base de données capable de stocker des chaines, de valeurs et des données pour la manipulation simple et centralisée de la configuration.
EET	Un format flexible de récipient pour stocker des images binaires et des données.
Edje	Une bibliothèque d'abstraction d'image et un ensemble d'outils basé sur EVAS, permettant de séparer complètement l'aspect de l'interface utilisateur du code de l'application grace à la transmission de signal.
Embryo	Un langage de script typiquement utilisé avec Edje pour un contrôle avancé.
Etox	Une bibliothèque de formatage et de manipulation de texte ayant des possibilités de stylisations.
Esmart	Une bibliothèque composée de divers objets EVAS intelligents prévus pour une réutilisation facile, incluant par exemple la populaire transparance.
Epeg	Une bibliothèque de création d'images JPEG miniatures rapide comme l'éclair indépendante des autres composants d'EFL.
Epsilon	Une bibliothèque de création d'images miniatures flexible et rapide prennant en charge PNG, XCF, GIF et JPEG.
Evoak	Une bibliothèque de serveur de canvas EVAS incluant un ensemble d'outils.
EWL	Une bibliothèque complète de widget.
Emotion	Une bibliothèque d'objets Evas pour la vidéo et la lecture de DVD qui utilise libxine.

Chapitre 2. Imlib2

Imlib2 est le successeur d'Imlib. Ce n'est pas seulement une version plus récente - c'est une bibliothèque complètement nouvelle. Imlib2 peut être installé parallèlement à Imlib puisque ce sont effectivement deux bibliothèques différentes - bien qu'elles aient des fonctionnalités similaires.

Imlib2 est capable de faire tout ce qui suit:

- Charger des images depuis un ou plusieurs formats
- Sauvegarder des images dans un ou plusieurs formats
- Faire des rendus d'image sur d'autres images
- Faire des rendus d'image sur une surface d'affichage X-Windows
- Produire des pixmaps et des masques pixmap d'images
- Appliquer des filtres aux images
- Appliquer des rotations aux images
- Accepter les données RGBA pour les images
- Redimensionner les images
- Faire des mélanges Alpha d'images avec d'autres images ou des surfaces d'affichage
- Appliquer une correction de couleur, des tables de modification et des facteurs aux images
- Faire des rendus d'image sur d'autres images avec correction de couleur et tables de modification
- Faire des rendus de texte truetype anti-aliasé
- Faire des rendus de texte truetype anti-aliasé dans n'importe quel angle
- Faire des rendus de lignes anti-aliasées
- Faire des rendus de rectangles
- Faire des rendus de dégradés multicolor linéaires
- Mettre intelligement les données en cache pour des performances maximum
- Allouer les couleurs automatiquement
- Permettre le contrôle complet sur le cache et l'allocation de couleur
- Fournir un code MMX hautement optimisé pour les routines de base
- Fournir une interface de filtre plug-in
- Fournir le chargement d'image et l'interface de sauvegarde à la volée
- Etre la bibliothèque de composition d'image, de rendu et de manipulation la plus rapide pour X

Si ce que vous voulez faire n'apparait pas quelque part dans la liste ci-dessus, alors il est fort probable qu'Imlib2 ne le fasse pas. Sinon, il le fera surement plus rapidement que n'importe quelle autre bibliothèque que vous pourriez trouver (en incluant gdk-pixbuf, gdkrgb, etc...) principalement en raison d'un code hautement optimisé et d'un intelligent sous-ensemble qui effectu le sale travail pour vous et qui assemble les morceaux à votre place. Ainsi, vous pouvez faire le paresseux et laisser Imlib2 faire toutes les optimisations.

Imlib2 fournit un puissant moteur pour la manipulation et le rendu d'image. En utilisant les chargeurs il peut supporter toute une variété de formats comprenant BMP, GIF (via unGIF), JPEG, PNG, PNM, TGA, TIFF, XPM et plus encore.

Recette: Filigranage d'Image

Ben technikolor Rockwood <benr@cuddletech.com>

Avec tant d'individus mettant tant d'images en ligne, il devient facile d'oublier d'où elles proviennent et difficile de s'assurer que du matériel contenant un copyright ne soit pas mal utilisé par inadvertance. Ajouter seulement une image filigranée, telle que le logo de vos sites, à chacune de vos images peut vous permettre de résoudre ces deux problèmes. Mais, ajouter à la main les filigranes peut vite devenir long et répétitif. Imlib2 peut facilement être employé à résoudre ce problème. Il suffit de choisir une image d'entrée, de spécifier l'image qui servira de filigrane (votre logo), de positionner le filigrane par rapport à l'image d'entrée et de sauvegarder le tout en une nouvelle image que l'on pourra utiliser sur son site. L'application devrait ressembler à ceci:

Exemple 2.1. Programme de Filigranage 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);
}
```

Regardons cet exemple: nous faisons tout d'abord une vérification basique des arguments, acceptant une image d'entrée comme premier argument et un nom pour l'image de sortie qui sera notre copie filigranée. En utilisant `imlib_load_image()` nous chargeons l'image d'entrée et prenons ses dimensions en utilisant les fonctions get. Avec la fonction `imlib_clone_image()` nous pouvons créer une copie de l'image d'entrée, qui sera la base de notre image finale. Ensuite nous chargeons l'image filigrane. Notez qu'alors nous utilisons

`imlib_context_set_image()` pour changer le contexte et passer de l'image d'entrée (image_input) à l'image filigrane (image_watermark). Nous prenons également ses dimensions. Dans le bloc final nous faisons deux calculs simples pour déterminer le positionnement du filigrane sur l'image finale, dans le cas présent nous voulons qu'il soit dans de coin inférieur droit. La fonction magique qui effectue vraiment le travail dans ce programme est `imlib_blend_image_onto_image()`. Notez que nous changeons encore de contexte en passant à l'image de sortie avant celà. La fonction de blend permet, comme son nom le suggère, de mélanger deux images que nous appelerons image source et image destination. Cette fonction mélange une image source à l'image du contexte dans lequel on est, l'image destination. Les arguments passés à `imlib_blend_image_onto_image()` peuvent sembler compliqués : nous devons lui dire quelle source employer (le filigrane), si nous voulons fusionner le canal alpha (0 pour non), donner les dimmensions de l'image source (x, y, w, h) et les dimensions de l'image destination (x, y, w, h). Vous noterez que dans cet exemple les positions de x et y de l'image source (filigrane) sont égales à 0 et que nous gardons ses dimensions intactes. La destination (l'image d'entrée) est placée dans le coin inférieur droit moins les dimensions du filigrane, et ensuite nous donnons la largeur et la hauteur du filirgane. Enfin, nous utilisons la fonction `imlib_save_image()` pour sauvegarder l'image finale.

Même si cet exemple devrait sensiblement être amélioré pour un usage réel, il décrit les bases du blending avec Imlib2 afin de résoudre efficacement un problème très commun.

Recette: Redimensionnement d'Image

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

Alors que de plus en plus de gens mettent des images sur Internet, il arrive souvent de vouloir réduire les dimensions de ces images afin de gagner de la bande passante. Ceci peut être facilement fait grâce à un simple programme Imlib2.

Cette recette prend en compte le nom de l'image d'entrée, la largeur désirée, la hauteur désirée, le nom de l'image finale et redimensionne l'image d'entrée aux dimensions données, sauvegardant le résultat, l'image finale.

Exemple 2.2. Redimensionnement d'Image

```
#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;
}
```

Avec cet exemple nous commençons par une vérification minimale des arguments, en s'assurant seulement que nous en avons le bon nombre.

L'image source est chargée en appelant la fonction `imlib_load_image()` qui charge les données de l'image en mémoire. Si l'appel échoue, la fonction retournera NULL. Une fois que nous avons les données de l'image, nous devons faire en sorte que cette image devienne notre contexte courrant. Ceci permet à Imlib2 de savoir sur quelle image les opérations seront effectuées. Nous faisons pour celà appel à `imlib_context_set_image()`. Nous pouvons ensuite procéder au redimensionnement grâce à la fonction `imlib_create_cropped_scaled_image()` qui prend comme arguments les positions de départ x et y, la largeur et la hauteur d'origine et la largeur et la hauteur de redimensionnement. La raison pour laquelle nous donnons les informations originelles est que cette fonction peut aussi retailler une image si on le désire. Pour cela il suffit de modifier x et y ainsi que la largeur et la hauteur d'origine en fonction de ses besoins. Le résultat étant une nouvelle image : out_img. Si jamais le redimensionnement échoue, la fonction retournera NULL. Nous changeons ensuite le contexte en passant à out_img et finissons par `imlib_save_image()` qui sauverdera notre image.

Bien que cet exemple soit simple, il montre néanmoins la simplicité du redimensionnement d'image à l'aide de l'API Imlib2.

Recette: Rotation Libre

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

Il arrive parfois que l'on désire appliquer une rotation à une image avec un angle particulier. Imlib2 rend ceci très facile à faire. Cet exemple cherche à montrer comment s'y prendre. Si vous voulez appliquer une rotation de 90 degrés à une image, référez vous à la recette correspondante, car la recette ici présente laissera une bordure noire autour de votre image.

Exemple 2.3. Rotation Libre

```
#define X_DISPLAY_MISSING

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

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

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

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

    angle = (atof(argv[2]) * (M_PI / 180.0));
```

```
    out_img = imlib_create_rotated_image(angle);
    if (!out_img) {
        fprintf(stderr, "Failed to create rotated image\n");
        return 1;
    }

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

Après avoir une simple vérification d'argument nous passons au travail d'Imlib2. On commence par charger l'image scpécifiée dans la mémoire avec `imlib_load_image()` en donnant le nom de l'image comme paramètre. On peut alors prendre cette image et la placer dans le contexte courrant avec `imblib_context_set_image`. Les contextes sont utilisés par Imlib2 afin de savoir sur quelle image on travaille. A chaque fois que vous voulez faire appel à une fonction Imlib2 sur une image, celle-ci doit être spécifiée comme étant le contexte courrant. Ensuite nous convertissons l'angle donné en degrés en radians car la fonction de rotation Imlib2 fonctionne en radians. La rotation est effectuée par `imlib_create_rotated_image()`. La fonctione de rotation retournera alors la nouvelle image. Afin de la sauvegarder nous devons la placer dans le contexte, toujours avec `imlib_context_set_image()`. Finalement, un simple appel à `imlib_save_image()` avec le nom de l'image modifiée sauvegarde la nouvelle image.

La fonction de rotation d'Imlib2 placera une bordure noire autour de l'image afin de remplir l'espace vide. Cette bordure est calculée de façon à ce que l'image issue de la rotation remplisse le cadre de sortie. Cela causera l'apparition de bordures autour de l'image finale même dans le cas d'une rotation à 180 degrés.

Recette: Rotation de 90 degrés

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

Avec un appareil photo numérique, il est souvent souhaitable d'effectuer une rotation d'image de: 90, 180 ou 270 degrés. Cette recette montrera comment faire ceci simplement avec Imlib2. Notez qu'avec cette recette vous n'aurez pas de burdures noires autour des images comme dans l'exemple de rotation libre.

Exemple 2.4. 90 degree Image rotation

```
#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;
```

```
}
```

Après une vérification d'erreur minimale nous chargeons l'image à laquelle on veut appliquer la rotation avec un appel à `imlib_load_image()`. Cette fonction accepte un nom de fichier et retourne l'objet Imlib_Image, ou NULL en cas d'erreur. Une fois que cette image est chargée nous la plaçons dans le contexte courrant, afin qu'Imlib2 effectue ses opérations sur celle-ci, avec `imlib_context_set_image()`. La rotation se fait grâce à `imlib_image_orientate()`. Le paramètre à _orientate fait varier le taux de rotation. Les valeurs possibles sont: [1, 2, 3] correspondant respectivement à une rotation en sens horaire de [90, 180, 270] degrés. Enfin, nous appelons `imlib_save_image()` en lui donnant le nom de l'image finale afin qu'Imlib2 sauvergarde l'image à laquelle on a appliquer la rotation.

Avec cet exemple entre vos mains vous devriez être capable d'effectuer des rotations d'images par intervales de 90 degrés grâce à Imlib2.

Recette: Renversement d'Image

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

Imlib2 contient des fonctions permettant de renverser une image. Ceci pouvant être fait horizontalement, verticalement ou diagonalement. Cette recette montrera comment mettre en application cette fonctionnalité.

Exemple 2.5. Renversement d'Image

```
#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;
}
```

Cet exemple commence par une vérification d'erreur minimale puis nous chargeons l'image d'entrée grâce à la fonction `imlib_load_image()`, en passant le nom du fichier à charger. `imlib_load_image()` retournera un objet Imlib_Image, ou NULL en cas d'erreur. Une fois que nous avons cet objet image, nous le plaçons dans le contexte courrant avec `imlib_context_set_image()`. Ceci indique à Imlib2 que c'est sur cette image que nous voulons travailler et que toutes les opérations Imlib2 seront appliquées à celle-ci. Ensuite nous décidons du type de renversement que nous voulons faire. Et nous l'appliquons en appelant `imlib_image_flip_horizontal()`, `imlib_image_flip_vertical()`, ou `imlib_image_flip_diagonal()`. Le renversement diagonal prend essentiellement le coin supérieur gauche pour le mettre dans le coin inférieur droit. Le coin supérieur droit devenant alors le coin inférieur gauche. Une fois l'image renversée, on appelle `imlib_save_image()` en lui donnant le nom du nouveau fichier et c'est finit.

Ceci vous donne un aperçu du renversement d'image avec Imlib2. Certains perfectionnements seront nécessaire avant d'intégrer cet exemple à une réelle application mais la base est là.

Chapitre 3. EVAS

Evas est une API de canvas matériellement accélérée pour X-Windows pouvant afficher du texte anti-aliasé, des images super et sous-échantillonnées, des mélanges alpha, mais aussi utiliser les primitives X11 concernant les pixmaps, les lignes et les rectangles pour plus de vitesse si votre CPU ou votre matériel graphique est trop lent.

Evas n'a pas besoin qu'on lui donne beaucoup d'informations sur les caractéristiques d'affichage de votre XServer, il les connait. Tout ce que vous devriez lui dire est combien de couleurs (au maximum) utiliser si il ne s'agit pas d'un affichage truecolor. Par défaut il est suggéré d'utiliser 216 couleurs (ceci étant égal à un cube en couleur de 6x6x6 - exactement le même cube coloré que Netscape, Mozilla, gdkrgb et d'autres utilisent, ainsi les couleurs seront partagées). Si Evas ne peut pas assigner assez de couleurs il réduira la taille du cube en couleur jusqu'à ce qu'il atteigne le noir et blanc. Ainsi, il peut afficher sur n'importe quoi depuis un terminal en noir et blanc à du VGA 16 couleurs, du 256 couleurs et plus, jusqu'à des couleurs en 15, 16, 24 et 32 bit.

Recette: Utiliser Ecore_Evas pour simplifier l'initialisation de canvas X11

Ben technikolor Rockwood

Evas est une bibliothèque puissante et simple d'utilisation, mais avant de pouvoir un canvas une surface d'affichage X11 doit être configurée. Configurer manuellement X11 peut être une tâche longue et frustrante qui déconcentre de ce que l'on veut vraiment faire: développer une application Evas. Mais tout ceci peut être évité en utilisant le module Ecore_Evas d'Ecore pour effectuer le dur travail pour vous.

L'exemple qui suit est un modèle de base qui peut servir de point de départ pour toute application Evas que vous voudriez construire, réduisant ainsi de manière significative le temps d'élaboration.

Exemple 3.1. Modèle 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;
}
```

Tout les détails au sujet de Ecore_Evas se trouvent au chapitre Ecore de ce livre, mais ce model basic vous permettra de jouer directement avec Evas. Les appels importants sont `ecore_evas_borderless_set()` qui définit si la fenêtre Evas est fenêtrée par votre window manager ou sans bordure, et `evas_font_path_append()` qui définit le chemin vers les polices employées par votre application Evas.

Recette: Raccourcis clavier, grace aux événements clavier EVAS

Ben technikolor Rockwood

Beaucoup d'applications tirent bénéfice de fournir des raccourcis clavier pour les opérations courrantes. Que ce soit pour accepter du texte d'une façon dont EFL n'a pas l'habitude ou bien juste pour associer la touche + à l'augmentation du volume d'un mixer, les raccourcis clavier peuvent être le petit plus qui fera de votre application un hit.

Le code qui suit est une application simple et complète dont l'intérêt est d'explorer les raccourcis clavier en utilisant les retours d'événements EVAS. On crée ici une fenêtre noire de 100x100 pixels dans laquel on peut frapper les touches du clavier.

Exemple 3.2. Saisie de touches avec les événements 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;
}
```

Vous pouvez compiler cet exemple de la manière suivante:

Exemple 3.3. Compilation

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

Dans cet exemple le canvas est configuré de façon usuelle grace à Ecore_Evas qui fait le sale boulot. La magie opère par l'action du callback `evas_object_event_callback_add()`.

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

En ajoutant un callback à base_rect, qui est en fait le fond du canvas, nous pouvons executer une fonction (`key_down()` dans ce cas) à chaque fois que l'on rencontre un événement, définit dans `Evas.h` comme EVAS_CALLBACK_KEY_DOWN.

Il y a une chose très importante à faire en plus de définir un callback: paramétrer le focus. La fonction `evas_object_focus_set()` mets le focus sur un objet Evas donné. C'est l'objet qui a le focus qui acceptera réellement les événements, même si vous avez explicitement définit l'objet Evas auquel le callback est attaché. Notez également que deux objets ne peuvent pas avoir le focus en même temps. Le problème le plus courrant avec les callbacks Evas est l'oubli du réglage de focus.

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

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

La fonction `key_down()` est appelée à chaque fois qu'une touche est pressée. La déclaration de cette fonction est celle d'un callback standard Evas (cf `Evas.h`). L'information importante que l'on cherche à connaitre est la touche qui a été pressée, contenue dans la structure Evas `event_info`. Après avoir paramétrer la structure `Evas_Event_Key_Down` comme ci-dessus, nous pouvons accéder à l'élément `keyname` afin de déterminer quelle touche a été pressée.

Dans la plupart des cas vous utiliserez probablement `switch` ou `if` pour définir quelle touche fait quoi et il est recommandé de coupler cette fonctionnalité avec une configuration EDB afin de fournir une centralisation et une expansion facile du paramétrage des raccourcis clavier de votre application.

Recette: Introduction aux objets intelligents Evas

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

En prennant l'habitude de travailler avec Evas, vous devriez avoir de plus en plus d'`Evas_Objects` sur lesquels vous travaillez et appliquez les mêmes opérations afin des garder synchronisés. Il serait beaucoup plus pratique de grouper tous ces `Evas_Objects` en un seul objet auquel les transformations seront appliquées.

Les objets intelligents Evas donnent la possiblité d'écrire vos propres objets et de voir Evas appeler vos fonctions pour effectuer le déplacement, le redimensionnement, le masquage et toutes les autres choses dont un `Evas_Object` est responsable. Avec les callbacks `Evas_Object`, les objets intelligents vous permettent de définir vos propres fonctions afin que l'objet supporte n'importe quelle opérations que vous pourriez lui demander.

Cett introduction est divisée en trois fichiers: foo.h, foo.c, et main.c. L'objet intelligent créé s'appelle `foo` et est définit dans les fichiers foo.[ch]. main.c est là pour montrer comment un objet intelligent peut être utilisé.

L'objet intelligent en lui-même est en fait deux carrés, l'un à l'intérieur de l'autre, dont celui du centre est espacé de 10% par rapport aux bords du carré externe. Pendant que le programme principal s'exécute un callback Ecore repositionne et redimensionne l'objet intelligent.

Cet objet intelligent est tiré d'un modèle écrit par Atmos: `www.atmos.org/code/src/evas_smart_template_atmos.c` `[http://www.atmos.org/code/src/evas_smart_template_atmos.c]` qui à son tour est tiré d'un modèle écrit par Rephorm.

D'abord nous devons définir l'interface externe de notre objet intelligent. Nous avons donc besoin d'un appel pour créer le nouvel objet.

Exemple 3.4. foo.h

```
#ifndef _FOO_H_
#define _FOO_H_

#include <Evas.h>

Evas_Object *foo_new(Evas *e);

#endif
```

Une fois que c'est fait, nous entrons dans le ventre de la bête, le code de l'objet intelligent.

Exemple 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;
};
```

L'objet `Foo_Object` contiendra toutes les informations dont nous avons besoin. Dans ce cas il s'agit du carré extérieur, du carré intérieur, d'un objet pour le clipping et la position courrante et la taille de l'objet.

```
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);
```

Les prédéclarations requises pour l'objet intelligent. Elles seront expliquées lorsque nous en viendrons à leur implémentation.

```
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()`est notre interface externe et est responsable de la mise en place de l'objet en lui-même. L'appel de `foo_object_new()` fera le plus gros de la création d'objet. `evas_object_smart_data_get()` est plus une vérification d'erreur qu'autre chose. Lorsque `foo_object_new()` est en cours d'exécution cela ajoutera l'objet intelligent à l'objet evas et il en résultera un appel `add` à l'objet. Dans notre cas, cet appel créera un `Foo_Object`. Ainsi, nous nous assurons juste qu'un `Foo_Object` a été créé.

```
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;
}
```

Notre fonction `foo_object_new()` a la simple tâche d'ajouter notre objet intelligent à l'objet Evas donné. Ceci est fait grace à `evas_object_smart_add()` en lui passant en paramètres l'Evas et l'objet `Evas_Smart *`. Notre `Evas_Smart *` est produit par l'appel de `_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;
}
```

Vous noterez que dans cet fonction `Evas_Smart *smart` est déclaré comme étant `static`. C'est parce que peu importe le nombre d'`Evas_Objects` que l'on crée, il n'y aura toujours qu'un objet `Evas_Smart`. Comme Raster l'a mentionné, un `Evas_Smart` est comme une définition de classe C++, pas comme une instance. L'`Evas_Object` est une instance de l'`Evas_Smart`.

L'objet intelligent en lui-même est créé par l'appel de `evas_smart_new()`. Nous passons à cette fonction le nom de l'objet intelligent, toutes les routines callback pour cet objet et n'importe quelles données utilisateur. Dans ce cas il n'y a pas de données utilisateur donc nous passons 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);
}
```

Lorsque nous appelons `evas_object_smart_add()` à l'intérieur de `foo_object_new()`, cette fonction, `_foo_object_add()` sera appelée de façon à ce que nous puissions paramétrer n'importe quelle donnée interne pour l'objet intelligent.

Pour cet objet intelligent nous avons paramétré trois `Evas_Objects` internes. `data->clip` est utilisé pour clipper les deux autres objets, `data->outer` est notre rectangle externe et `data->inner` notre rectangle interne. Les rectangles interne et externe sont montré immédiatement. L'objet clip lui ne sera montré que lorsque l'utilisateur aura appelé `evas_object_show()` sur cet objet.

Finallement nous appelons `evas_object_smart_data_set()` pour spécifier notre nouveau `Foo_Object` comme une donnée de cet objet intelligent. Cette donnée sera rapartiée dans les autres fonctions de cet objet en appelant `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);
    }
}
```

Le callback `_foo_object_del()` sera exécuté si l'utilisateur appelle `evas_object_del()` sur notre objet. Pour cet objet c'est aussi simple que d'effacer nos trois rectangles et de libérer notre structure de données `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);
    }
}
```

Ce groupe de fonctions: `_foo_object_layer_set()`, `_foo_object_raise()`, `_foo_object_lower()`,

`_foo_object_stack_above()`, et `_foo_object_stack_below()` fonctionnent toutes de la même façon, en appliquant la fonction requise `evas_object_*` à l'objet `data->clip`.

Ces fonctions sont déclenchées respectivement par l'utilisation de: `evas_object_layer_set()`, `evas_object_raise()`, `evas_object_lower()`, `evas_object_stack_above()`, et `evas_object_stack_below()`.

```
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;
    }
}
```

Le callback `_foo_object_move()` est déclenché lorsque `evas_object_move()` est appelé pour notre objet. Chacun des objets internes est déplacé vers sa position correcte grace à des appels à `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;
    }
}
```

Le callback `_foo_object_resize()` sera déclanché lorsque l'utilisateur appellera `evas_object_resize()` pour notre objet. Ainsi, nous devons redimensionner `data->clip` et `data->outer` aux dimensions de notre objet. Nous pouvons le faire avec des appels à `evas_object_resize()`. Nous pouvons ensuite déplacer et redimensionner l'objet `data->inner` afin qu'il reste dans la bonne position par rapport à notre rectangle extérieur. Ce que nous faisons respectivement grace à `evas_object_move()` et `evas_object_resize()`. Alors nous enregistrons la largeur et la hauteur courrante dans notre objet afin de pouvoir les rappeler par la suite.

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

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

Le callback `_foo_object_show()` sera déclenché lorsque `evas_object_show()` sera appelé pour notre objet. Pour afficher notre objet, tout ce que nous devons faire est afficher la région clip puisque nos rectangles sont clippés par elle. Nous le faisons en appelant `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);
}
```

Le callback `_foo_object_hide()` sera déclenché lorsque `evas_object_hide()` sera appelé pour notre objet. Comme nous utilisons un objet interne de clipping, nous devons juste cacher l'objet clip, `data->clip`, afin de cacher notre objet intelligent. Nous le faisons en appelant `evas_object_hide()`.

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

La fonction `_foo_object_color_set()` sera lancée si `evas_object_color_set()` est appelé pour notre `Evas_Object`. Mais, comme nous ne voulons pas ici de changement de couleurs, nous ignorons ce 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);
}
```

Le callback `_foo_object_clip_set()` sera déclenché lorsqu'un appel à `evas_object_clip_set()` est fait pour notre objet. Dans ce cas nous propageons ce paramètre à notre objet interne, data->clip grace 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);
}
```

Le callback `_foo_object_clip_unset()` sera déclenché lorsqu'un appel à `evas_object_clip_unset()` est fait pour notre objet. Nous enlevons le paramètre de clip de notre objet interne avec `evas_object_clip_unset()`.

Une fois que le code d'objet intelligent est finit, nous pouvons créer notre programme principal pour utiliser ce nouvel objet.

Exemple 3.6. main.c

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

#define WIDTH 400
#define HEIGHT 400
#define STEP 10
```

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

static int timer_cb(void *data);

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

    ecore_init();

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

    evas = ecore_evas_get(ee);

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

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

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

    return 0;
}

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

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

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

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

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

La plus grande partie de ce programme est similaire à celui de la recette utilisant Ecore_Evas vu plus haut. La création de notre nouvel objet intelligent est contenu dans cette portion de code:

```
    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);
```

Une fois que `foo_new()` a retouné notre objet nous pouvons le manipuler avec des appels Evas normaux, ainsi nous spécifions sa position, sa taille, sa couche et affichons l'objet.

Maintenant que notre objet intelligent est créé et affiché, nous paramétrons un temporisateur Ecore afin qu'il se déclenche toutes les 0.1 secondes. Lorsque le temporisateur est déclenché il execute la fonction `timer_cb()`. Ce callback rétrécira ou accroissera la taille de notre objet intelligent tout en le centrant dans notre fenêtre principale.

Compiler cet exemple est aussi simple que:

Exemple 3.7. Compilation

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

Les objets intelligents Evas sont simples à mettre en oeuvre mais fournissent un puissant méchanisme d'abstraction de parties de votre programme. Si vous souhaitez en savoir plus sur les objets intelligents, jettez un coup d'oeil à la section Esmart, Etox ou Emotion.

Chapitre 4. Ecore

Qu'est-ce qu'Ecore? Ecore est la couche principale d'abstraction d'événement et la couche d'abstraction de X qui permet de faire des sélections, du Xdnd, des actions en rapport avec X plus générales et de manipuler les boucles d'événements, les arrêts et l'attente avec rapidité, optimisation et commodité. Comme c'est une bibliothèque séparée, tout le monde peut se servir d'Ecore pour se rendre le travail plus facile.

Ecore est compètement modulaire. On trouve à sa base les manipulateurs et timers d'événements, et des fonctions d'initialisation et d'arrêt. Les modules d'abstraction pour Ecore incluent:

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

Ecore est tellement modulaire et puissant qu'il peut être très utile même pour de la programmation non graphique. Par exemple, plusieurs serveurs web ont été écrit en se basant uniquement sur Ecore et le module Ecore_Con pour la communication socket.

Recette: Introduction à Ecore Config

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

Le module Ecore_Config fournit au programmeur un moyen simple de mettre ne place des fichiers de configuration pour ses programmes. Cette recette donnera un exemple de la façon d'intégrer la base d'Ecore_Config à votre programme et de l'utiliser pour obtenir les données de configuration.

Exemple 4.1. Programme Simple Ecore_Config

```
#include <Ecore_Config.h>

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

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

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

    ecore_config_load();

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

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

```
    free(str);

    ecore_config_shutdown();
    return 0;
}
```

Comme vous pouvez le voir dans cet exemple, l'utilisation de base d'Ecore_Config est simple. Le système est initialisé grâce à un appel à `ecore_config_init`. Le nom de votre programme définit où Ecore_Config devra rechercher votre base de données de configuration. Le nom du répertoire et du fichier sont: ~/.e/apps/PROGRAM_NAME/config.db.

Pour chaque variable de configurationque vous obtenez d'Ecore_Config, vous pouvez assigner une valeur par défaut dans le cas où l'utilisateur n'aurait pas de fichier config.db. Ces valeurs par défaut sont assignées avec ecore_config_*_default où * est un des types Ecore_Config. Le premier paramètre est la clef sous laquel on y aura accès. Ces clefs doivent être uniques au sein de votre programme. La valeur passée doit être du type approprié pour cet appel.

L'appel à `ecore_config_load` lira les valeurs du fichier config.db. Après quoi nous pouvons accéder aux fichiers grâce aux méthodes ecore_config_*_get (* est encore le type de donnée désiré). Ces routines prennent le nom de la clef d'un élément et renvoient la valeur liée à cette clef. Chaque fonction renvoie un type correspondant à son nom d'appel.

`ecore_config_shutdown` est alors appelée pour arrêter le système Ecore_Config avant que le programme finisse.

Exemple 4.2. Compilation

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

Pour compiler ce programme vous pouvez utiliser le script ecore-config pour obtenir toutes les informations de liens et de bibliothèques nécessaires à Ecore_Config. Si vous lancez ce programme tel quel, vous afficherez les valeurs par défaut données à ecore_config. Une fois que vous savez que votre programme fonctionne, vous pouvez créer un fichier config.db afin d'en lire les valeurs.

Exemple 4.3. Script simple config.db (build_cfg_db.sh)

```
#!/bin/sh

DB=config.db

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

Lorsque build_cfg_db.sh est exécuté, il crée un fichier config.db dans le dossier courant. Ce fichier peut alors être copié dans ~/.e/apps/PROGRAM_NAME/config.db où PROGRAM_NAME est la valeur passée à ecore_config_init. Une fois que ce fichier est en place, l'exécution du programme de test affichera les valeurs du fichier de configuration et non celles par défaut. Vous pouvez spécifier autant de valeurs qu'il y a de clefs dans votre fichier de configuration et Ecore_Config les affichera.

Recette: Auditeurs Ecore Config

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

Lorsque l'on utilise Ecore_Config pour manipuler la configuration d'un application, il est intéressant de pouvoir savoir lorsque la configuration a changé. Ceci peut être fait grâce à l'utilisation d'auditeurs au sein d'Ecore_Config.

Exemple 4.4. Audit Ecore_Config

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

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

enum {
    EX_ITEM,
    EX_STR_ITEM,
    EX_FLOAT_ITEM
};

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

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

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

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

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

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

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

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

        case EX_STR_ITEM:
```

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

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

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

On commence par mettre en place Ecore_Config comme d'habitude et on crée quelques clefs par défaut. Les parties intéressantes arrivent avec l'appel à `ecore_config_listen()`. C'est l'appel qui dit à Ecore_Config que nous voulons être informés en cas de changements de configuration. `ecore_config_listen()` prend cinq paramètres:

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

Le champ `name` est une chaîne que l'on donne pour identifier ce callback d'audit. La `key` est le nom de la clef que l'on veut écouter, qui sera donc le même que le nom de clef donné dans les appels _default plus haut. Le `listener callback` est la fonction à exécuter en cas de changement. L'`id tag` est une étiquette qui peut être donnée à chaque auditeur et qui sera passée à la fonction de callback. Pour finir, les `user data` sont les données que vous souhaitez voir passées au callback lorsqu'il est exécuté.

La fonction de callback a une signature semblable à:

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

`key` est le nom de la clef que l'on audit. Le paramètre `type` contient le type d'Ecore_Config. Il peut faire partie de:

PT_NIL	Sans valeur
PT_INT	Type Entier
PT_FLT	Type Flottant
PT_STR	Type Chaîne
PT_RGB	Type Couleur
PT_THM	Type Theme

Le paramètre `tag` est la valeur donnée plus haut lors de la création de l'auditeur. Enfin, `data` correspond à n'importe quelles données utilisateur attachées à l'auditeur lorsqu'il a été créé.

Si vous voulez retirer l'auditeur plus tard, `ecore_config_deaf()` vous le permet. Il prend trois paramètres:

- name
- key
- listener callback

Chacun de ces paramètres correspond au paramètre donné à l'appel initial `ecore_config_listen()`.

Exemple 4.5. Compilation

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

Si vous lancez le programme, vous verrez les valeurs par défaut s'afficher sur votre écran. Si vous lancez examine comme suit:

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

(foo est le nom donné à `ecore_config_init()`). Vous devriez être capable de changer les réglages de votre application et, après avoir pressé 'save', de voir les valeurs modifiées s'afficher sur votre console.

Recette: Introduction à Ecore Ipc

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

La bibliothèque Ecore_Ipc fournit un wrapper robuste et efficace autour du module Ecore_Con. Ecore_Ipc vous permet de mettre au point vos communications serveur et manipule tous les trucs compliqués en interne. Cette recette vous donnera un exemple simple d'un client et d'un serveur Ecore.

Lorsqu'on travaille avec Ecore_Ipc, que l'on écrive une application client ou serveur, un créera un objet Ecore_Ipc_Server. Ceci est dû au fait que dans l'un ou l'autre des cas un serveur est en action, qu'on le mette en place ou qu'on communique avec. Après ça, tout est simple.

Exemple 4.6. Client Ecore_Ipc: préambule

```
#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);
```

Le fichier Ecore.h est inclus de façon à avoir accès au type de signal de sortie. Les fonctions seront expliquées plus tard lorsque leurs callbacks seront connectés.

Exemple 4.7. Client Ecore_Ipc: setup de 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);
```

Comme dit plus haut, même si nous écrivons une application client, nous utilisons un objet Ecore_Ipc_Server. Employer Ecore_Ipc exige l'initialisation d'Ecore lui-même. Nous le faisons en appelant `ecore_init`. Ecore_Ipc est alors initialisé avec `ecore_ipc_init`. Si l'un ou l'autre retourne 0, l'action appropriée est exécutée afin de défaire ce qui a été initialisé avant. Le callback ECORE_EVENT_SIGNAL_EXIT est connecté pour que nous puissions quitter avec élégance s'il y a lieu.

Exemple 4.8. Client Ecore_Ipc: création du client

```
    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);
```

Dans cet exemple nous créons une connection à distance au serveur nommé "localhost" sur le port 9999. Nous le faisons avec la méthode `ecore_ipc_server_connect`. Le premier paramètre est le type de connection que nous voulons établir, qui peut être un de ceux-ci: ECORE_IPC_REMOTE_SYSTEM, ECORE_IPC_LOCAL_SYSTEM, ou ECORE_IPC_LOCAL_USER. Si OpenSSL était disponible au moment de la compilation d'Ecore_Ipc, ECORE_IPC_USE_SSL peut être un autre type de connection, permettant de créer une connection SSL.

Les trois appels à `ecore_event_handler_add` mettent en place les callbacks pour les différents types d'événements que nous recevrons du serveur. Un serveur a été ajouté, un serveur a été supprimé, ou le serveur nous a envoyé des données.

Exemple 4.9. Client Ecore_Ipc: fin 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;
}
```

Dans le but de cet exemple, au départ le client envoie un message au serveur, auquel il répondra. Le message client est envoyé avec la commande `ecore_ipc_server_send`. `ecore_ipc_server_send` prend comme arguments le serveur vers lequel on envoie, l'opcode majeur du message, l'opcode mineur, la référence, la référence à, si une réponse est requise, les données et la taille des données. Ces paramètres, excepté le serveur sont à la guise du client

et peuvent se rapporter à ce que l'on veut. Ceci donne un maximum de flexibilité dans la création d'application client/serveur IPC.

Une fois que le message serveur est envoyé nous entrons dans la boucle ecore principale et attendons les événements. Si on quitte la boucle principale on supprime l'objet serveur, on arrête Ecore_Ipc avec un appel à `ecore_ipc_shutdown`, et on arrête ecore avec `ecore_shutdown`.

Exemple 4.10. Client Ecore_Ipc: sig_exit_cb

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

La fonction `sig_exit_cb` dit simplement à Ecore de quitter la boucle principale. Ce n'est pas strictement nécessaire car si on ne fait qu'appeler `ecore_main_loop_quit()`, Ecore s'en occupe de toutes façons si aucun traitement n'est définit. Mais ceci montre comment traiter un signal si vous en avez besoin pour votre application.

Exemple 4.11. Client Ecore_Ipc: les 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;
}
```

Ces trois callbacks, `handler_server_add`, `handler_server_del`, et `handler_server_data` sont le corps du client, ils traitent tous les événements en relation avec le serveur auquel on est connecté. Chacun des callbacks a une structure d'événement associée, Ecore_Ipc_Event_Server_Add, Ecore_Ipc_Event_Server_Del et Ecore_Ipc_Event_Server_Data, contenant les informations sur l'événement lui-même.

Lorsque nous nous connectons au serveur, le callback `handler_server_add` est exécuté, permettant ainsi l'accomplissement des réglages nécessaires.

Si le serveur intérompt la connection le callback `handler_server_del` est lancé afin de permettre le nettoyage requis.

Lorsque le serveur envoie des données au client le callback `handler_server_data` est ecécuté. Ce qui, dans cet exemple, ne fait qu'afficher des informations à propos du message en lui-même et du corps du message.

Et voilà pour le client. Le code en lui-même est très simple grâce à l'abstraction fournie par Ecore.

Exemple 4.12. Serveur Ecore_Ipc: préambule

```
#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);
```

Comme pour le client, l'en-tête Ecore.h est inclus pour avoir accès au signal de sortie. L'en-tête Ecore_Ipc.h est nécessaire aux applications qui utilisent la bibliothèque Ecore_Ipc. Les fonctions seront expliquées avec leur code.

Exemple 4.13. Serveur Ecore_Ipc: setup 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);
```

C'est la même chose que pour le setup client vu plus haut.

Exemple 4.14. Serveur Ecore_Ipc: création du serveur

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

A la différence du client, nous ajoutons pour le serveur un auditeur au port 9999 sur la machine "localhost" avec l'appel `ecore_ipc_server_add`. Ceci créera et nous renverra l'objet serveur. Ensuite nous préparons le traitement des signaux, la différence avec le client étant ici que nous voulons cette fois des événements CLIENT au lieu d'événements SERVER.

Exemple 4.15. Serveur Ecore_Ipc: fin de main

```
    ecore_main_loop_begin();
```

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

C'est encore identique à l'arrêt du client, moins l'envoie de données au serveur.

Exemple 4.16. Serveur Ecore_Ipc: sig_exit_cb

La fonction `sig_exit_cb` est elle aussi identique à ce que nous avons vu pour le client.

Exemple 4.17. Serveur Ecore_Ipc: les 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;
}
```

Les callbacks sont similaires à ceux rencontrés dans l'application client. La différence principale étant que les événements sont _Client_ et non plus _Server_.

Le callback add sert lorsqu'un client se connecte au serveur, et le callback del (son opposé) lorsque le client se déconnecte. Le callback data sert lorsqu'un client envoie des données au serveur.

A la fin du callback `handler_client_data` nous faisons appel à `ecore_ipc_client_send`. On envoie ainsi les données au client. Comme lorsqu'on envoie vers le serveur, les paramètres sont: le client auquel on envoie, l'opcode majeur, l'opcode mineur, la référence, la référence à, si une réponse est requise, les données et la taille des données.

Exemple 4.18. Ecore_Ipc: compilation

```
CC = gcc
```

```
all: server client

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

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

clean:
    rm server client
```

Comme avec les autres applications Ecore, il est très simple de compiler une application Ecore_Ipc. Tant que Ecore a été compilé avec Ecore_Ipc, la simple invoquation de la commande 'ecore-config --cflags --libs' ajoutera tous les paths des bibliothèques requises et les informations de liaison.

Nous l'avons vu avec cet exemple, Ecore_Ipc est une bibliothèque facile à utiliser pour créer des applications client/serveur.

Recette: Timers Ecore

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

Si vous devez avoir un callback déclenché à un instant spécifique, avec la possibilité de répéter le timer continuellement, alors `Ecore_Timer` est ce que vous cherchez.

Exemple 4.19. Timers Ecore

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

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

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

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

    ecore_main_loop_begin();
    ecore_shutdown();

    return 0;
}

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

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

La création des timers est aussi simple qu'un appel à `ecore_timer_add()`. Ceci renverra une structure `Ecore_Timer` en cas de succès ou NULL en cas d'échec. Dans le cas présent nous ignorons les valeurs de retour.

Les trois paramètres de cette fonction sont:

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

Le timeout correspond au nombre de secondes au bout desquelles ce timer expirera. Dans le cas de cet exemple on spécifie respectivement 1 et 0.5 secondes. La fonction de callback est celle qui sera exécutée lorsque le timer expirera et user_data correspond à n'importe quelles données à la fonction de callback.

Toutes les fonctions de callback ont la même signature `int callback(void *data)`. La valeur de retour du timer devrait être 0 ou 1. Si vous renvoyez 0, le timer expirera et ne sera *pas* relancé. Si vous renvoyez 1, le timer sera reporté pour se reexécuter dans autant de temps que spécifié par le timeout. Cela permet d'activer ou non le timer suivant les exigences de votre programme.

Si vous avez un timer que vous voulez enlever à un certain moment dans le futur, vous pouvez appeler `ecore_timer_del(Ecore_Timer *)`. Si la suppression réussit, la fonction renverra le pointeur, ou NULL si échec. Après avoir appelé la fonction de suppression, la structure `Ecore_Timer` ne sera plus valide et vous ne devriez plus l'utiliser dans votre programme.

Compiler cet exemple est aussi simple que:

Exemple 4.20. Compilation

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

Si vous lancez ce programme vous devriez voir une série de '1' et de '2' sur votre écran, avec deux fois plus de '2' que de '1'.

Les `Ecore_Timers` sont facile à mettre en oeuvre et à utiliser et fournissent un mécanisme puissant de synchronisation à vos programmes.

Recette: Ajout d'Evénements Ecore

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

Si jamais vous devez créer vos propres événements, vous pouvez facilement les lier au système d'événement d'Ecore. Ceci vous donne la capacité d'ajouter des événements dans la file d'attente et de voir Ecore les délivrer à une autre partie de l'application.

Le programme qui suit crée un événement et un timer. Quand le timer est déclenché il ajoute notre nouvel événement dans la file d'attente d'événement. Notre événement affichera alors un message et stoppera l'application.

Exemple 4.21. Exemple d'Evénement 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);
}
```

Chaque événement est associé à un identifiant. Cet identifiant est une simple valeur int qui est assigné par l'appel à `ecore_event_type_new()`. Une fois que nous avons cet identifiant, nous pouvons l'employer dans les appels à `ecore_event_handler_add()`. C'est tout ce qu'il y a à faire pour créer un événement.

Ecore nous donne la capacité d'attacher une structure d'événement à notre événement. Notez que vous devez faire attention, si vous ne spécifiez pas une fonction free pour la struct, Ecore utilisera une fonction générique qui appellera simplement `free` sur la valeur. Donc, n'attachez rien là-dedans dont vous ayez besoin sans faire attention. (Ou préparez vous à passer une soirée à pister d'étranges segv dans votre programme)

Dans cet exemple nous créons une simple `Event_Struct`. L'appel qui créera réellement l'événement est `ecore_event_add()`. Cette fonction prend en argument l'identifiant, n'importe quelles données événement, la fonction free pour notre événement, et n'importe quelles données à passer à la fonction free.

Donc, comme vous pouvez le voir, nous passons notre `Event_Struct` comme donnée d'événement et plaçons la fonction `event_free` pour nettoyer cette structure.

Et voilà. Vous pouvez compiler comme montré ci-dessous et tout devrait fonctionner.

Exemple 4.22. Compilation

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

Comme ceci le montre, il est vraiment facile d'étendre Ecore avec vos propre événements. Le système a été créé de façon à pouvoir être étendu à volonté.

Chapitre 5. EDB et EET

EDB est une librairie pratique de manipulation de bases de données basée sur la Berkeley DB 2.7.7 de Sleepycat Software. Elle est développée dans le but de rendre la manipulation d'informations de ces bases de données portable, simple, facile et efficace.

EET est une petite librairie dévoloppée dans le but d'écrire des blocs de données dans un fichier, et optionnellement, de compresser ces blocs à la volée (presque de la même façon que winzip), elle permet aussi des accès aléatoires rapides à un fichier. Elle ne zip pas étant donné que le zip lui même est plus complexe que cela n'est nécessaire, il fut seulement plus simple d'implementer cela ici.

EDB fournit une excélente méthode de stockage et de récupération de configurations d'applications, bien qu'elle puisse être utlisée pour bien plus que cela. Ebits, le prédécesseur de Edje, utilisais même EDB comment conteneur pour les thèmes Ebits avant EET. Un Edb consiste en une série de paires clef/valeurs, qui peuvent être de plusieurs types, dont numérique, flottant, chaines et données binaires. L'API simplifiée fournit des fonctions simples, complètes et unifiées pour gérer et accéder à votre base de données.

En plus de la librairie, une grande variétée d'outils est disponnibles pour accéder et modifier vos EDBs. L'outil edb_ed fournit une interface en ligne de commande simple pouvant être utilisée dans un script, cela est surtout utile pour une utilisation avec la suite GNU autotools. L'outil edb_vt_ed fournit une interface curses facile à utiliser. Enfin, edb_gtk_ed fournit une interface graphique élégante et simple, surtout utile pour l'utilisateur final pour éditer les données contenues dans des EDBs.

Eet est extrèmement rapide, petit et simples. Les fichiers Eet peuvent être vraiment petits et très compressés, les rendant optimisés pour l'envoi à travers l'internet sans devoir les archiver, compresser, décompresser ou installer. Ils autorisent des accès aléatoires en lecture à la vitesse de la lumière dès la création, les rendant parfait pour des données écrites une seules fois (ou rarement), mais lues très souvent, mais dont le programme ne souhaite pas les lire totalement dès le debut.

Il peut aussi encoder et décoder des structures de données en mémoire, aussi bien que des données d'images pour sauver de fichier Eet ou les envoyer à travers le réseau vers d'autres machines, ou simplement écrire dans des fichiers arbitraires sur le système. Toutes les données sont encodée à l'aide d'une méthode indépendante de l'architecture et peuvent être écrites et lues par n'importe quelle architecture.

Recette: Créer des fichiers EDB depuis la ligne de commande

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

Il est souvent nécessaire de créer des fichiers EDB depuis un simple script shell, cela peut avoir été une partie de procédure de construction.

Cela peut être facilement accompli en utilisant le programme **edb_ed**. **edb_ed**est une interface très simple pour EDB, autorisant création/edition/suppression de pairs clef/valeur dans les bases de données EDBs.

Exemple 5.1. Script shell de fichier EDB

```
#!/bin/sh

DB=out.db

edb_ed $DB add /global/debug_lvl int 2
edb_ed $DB add /foo/theme str "default"
edb_ed $DB add /bar/number_of_accounts int 1
```

```
edb_ed $DB add /nan/another float 2.3
```

Si le fichier de sortie (output) n'existe pas lors du premier appel à une commande `add` , le programme **edb_ed** créera le fichier et fera la configuration nécessaire. La fonction `add` est utilisée pour ajouter des entrées dans les bases de données. Le premier paramètre après add est la clé avec laquelle la donnée sera insérée dans la base de données. Cette clé sera utilisée par votre application pour récupérer la donnée dans le futur. Le paramètre suivant est le type de donnée à ajouter. Ce type peux prendre les valeurs suivantes :

- int
- str
- float
- data

Le dernier paramètre est la valeur qui sera associée à cette clé.

En utilisant le programme **edb_ed** vous pouvez rapidement et facilement créer/éditier/visionner n'importe quel fichier EDB requis par votre application.

Recette: Introduction à EDB

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

EDB fournit une interface de base de données puissant pour être utilisée dans votre application. Cette recette est une introduction simple qui va ouvrir une base de données, écrire plusieurs clés puis les lire.

Exemple 5.2. Introduction à 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;
}
```

Pour utiliser EDB vous devez inclure <Edb.h> dans votre fichier pour avoir accès à l'API. Les portions initiales du programmes sont quelques peu standards, j'ai une certaine tendance à faire des fautes de frappes, j'ai donc défini les différentes clés que j'utiliserais. Si nous avons un nom de fichier, nous essayons d'ouvrir/créer la base de données.

La base de données sera ouverte, ou si elle n'existe pas, créée avec l'appel à `e_db_open()` qui retournera NULL si une erreur à été rencontrée.

Dès que la base de données est ouverte nous pouvons écrire nos valeurs. Cela peux être fait à l'aide des trois appels : `e_db_int_set()`, `e_db_float_set()` et `e_db_str_set()`. Vous pouvez aussi créer des données génériques dans un fichier en utilisant `e_db_data_set()`.

Avec les données normales, vous pouvez sauver des méta-données concernant la base de données dans le fichier lui même. Ces données ne peuvent être récupérée à l'aide des methodes get/set traditionnelles. Ces propriétés sont configurée grâce à la fonction `e_db_property_set()`

Chaque méthode de configuration de type utilise trois paramètres :

- E_DB_File *db
- char *key
- value

Le paramètre `value` est du type correspondant à la méthode, int, float, char * ou void * pour _int_set, _float_set, _str_set et _data_set respectivement.

Une fois les valeurs enregistrées dans la base de données elles peuvent être lues à l'aide des méthodes "get". Chacune de ces méthodes requiert 3 paramètres et retourne un int. La valeur de retour est 1 en cas de succès, 0 dans les autres cas. Once the values are in the db they can be retrieved with the getter methods.

Comme pour les méthodes "set", les paramètres des méthodes "get" sont la base de donnéesn, la clé et un pointer vers l'endroit où placer la valeur.

Dès que nous avons finis avec la base de données nous pouvons la fermer avec un appel à `e_db_close()`. L'appel à `e_db_close()` ne nous garanti que la base de données à été écrite sur le dique, pour s'en assurer nous appelons `e_db_flush()` qui écrira toutes les bases de données non utilisées sur le disque.

Exemple 5.3. Compilation

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

Si vous exécutez le programme vous devriez voir les valeurs écrites sur votre écran, et après l'exécution il y aura un

fichier .db avec le nom que vous avez spécifié. Vous pouvez éditer le fichier .db à l'aide du programme **edb_gtk_ed** et voir les valeurs entrées.

Recette: Obtention de clé EDB

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

L'API EDB rend facile la récupération de toutes les clé disponibles dans une base de données. Ces clés peuvent ensuites être utilisées pour déterminer le type d'un objet dans la base de données, ou simplement récupérer un objet au besoin.

Exemple 5.4. Obtention de clé 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;
}
```

Récupérer les clés est fait trés simplement par l'appel de `e_db_dump_key_list()`. Cet appel retournera un tableau char ** de chaines de clés. Ces chaines, et le tableau lui même, doivent être libérées par l'application. `e_db_dump_key_list()` retournera aussi le nombre de clés dans la tableau dans le paramètre `num_keys`.

Vous noterez que nous n'avons pas besoin d'ouvrir la base de données pour l'appel `e_db_dump_key_list()`. Cette fonction travaille sur le fichier lui même plutôt que sur un objet base de données.

Exemple 5.5. Compilation

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

L'exécution du programme devrait produire un listing de toutes les clés dans la base données. Ce listing peut être vérifié par l'emploi d'un outil externe comme **edb_gtk_ed** pour lire la base de données.

Chapitre 6. Esmart

Esmart fournit à vos applications utilisant EVAS et EFL une variété d'objets EVAS intelligents qui donnerons à vos applications une puissance significative.

Recette: Introduction à Esmart Trans

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

La transparence devient de plus en plus un trait commun de nos applications. Dans ce but, l'objet Esmart_Trans à été créé. Cet objet fera tout le travail compliqué pour créer un fond transparent à votre application.

Esmart trans rend l'intégration d'un fond transparent à votre application très simple. Vous devez créér un objet trans, puis être sûrs de le mettre à jour lorsque la fenêtre est déplacée ou redimensionnée.

Exemple 6.1. Inclusions et déclarations

```
#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();
```

Chaque application utilisant un objet Esmart_Trans requiert les fichier en-tête Ecore, Ecore_Evas et Esmart/Esmart_Trans. Les quatres déclarations suivantes sont des callbacks d'ecore pour les évènements sur notre fenêtre, exit, delete, resize, et move respectivement. Les deux dernières déclaration sont des functions de simplifications utilisées dans notre exemple et ne sont pas indispensables dans notre programme.

Exemple 6.2. main

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

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

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

    ecore_evas_shutdown();
```

```
ECORE_SHUTDOWN:
    ecore_shutdown();

    return ret;
}
```

La routine principale pour cet exemple est relativement simple. Ecore et Ecore_Evas sont tous deux initialisés, avec une détection d'erreur appropriée. Nous créeons ensuite l'interface et entammons la boucle principale d'évènements ecore. Lorsque ecore exit nous fermons tout et renvoyant la valeur de retour.

Exemple 6.3. callbacks exit et del

```
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();
}
```

Ces callbacks exie et del sont des callbacks ecore génériques. Le callback exit n'est pas strictement nécessaire, étant donné que Ecore appellera `ecore_main_loop_quit()` if no handler is registered, but is included to show how its done.

Exemple 6.4. _freshen_trans

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

    if (!ee) return;

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

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

La routine `_freshen_trans` est une routine d'aide à la mise a jour de l'image sur laquelle s'applique la transparence. Elle sera appelée lorsque que nous nous aurons besoin de mettre à jour notre image vis à vis de ce qui est présent en dessous de l'image. La fonction récupère la taille courrante du ecore_evas, puis l'objet dans le nom est "bg" (ce nom est le même que celui que nous donnons à notre trans lorsque nous le créeons). Puis, tant que l'objet trans existe, nous demandons à esmart de raffraichir l'image étant affichée.

Exemple 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);
}
```

Lorsque la fenêtre est redimensionnée nous devons mettre à jour notre evas à la taille correcte puis mettre à jour l'objet trans pour afficher correctement le fond. Nous récupérons la taille courante de la fenêtre à l'aide de `ecore_evas_geometry_get` et la taille minimum/maximum de la fenêtre. Si la taille désirée est comprise dans les limites minimum/maximum pour notre fenêtre, nous recupérons l'objet "bg" (même titre une fois de plus) et le redimensionnons. Une fois le redimensionnement temriné, nous appelons la routine `_freshen_trans` pour mettre à jour l'image affichée en fond.

Exemple 6.6. move_cb

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

Lorsque la fenêtre est déplacée nous devons mettre à jour l'image affichée en transparence.

Exemple 6.7. Configuration de ecore/ecore_evas

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

    ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, sig_exit_cb, NULL);

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

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

    evas = ecore_evas_get(ee);
```

La première partie de make_gui est relative à la configuration de ecore et ecore_evas. Premièrement le callback exit

est attaché dans ECORE_EVENT_SIGNAL_EXIT, puis l'objet Ecore_Evas est créé avec le moteur logiciel X11. Le titre de la fenêtre est configuré et nous capturons les callbacks ci-dessus, deletre, resize et move. Finalement, nous récupérons l'evas pour le Ecore_Evas créé.

Exemple 6.8. Creation d'un objet 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);
}
```

Une fois que tout est paramétré nous pouvons créér l'objet trans. Le trans est créé par l'evas retourné par la fonction `ecore_evas_get`. Cette créaton initiale est faite par l'appel à `esmart_trans_x11_new`. Une fois qe nous avons l'objet, nous le déplacons pour qu'il débute à la position (0, 0) (le coin haut gouche), positionnons la couche à -5 et nomons l'objet "bg" (comme utilisé plus hautas used above). Puis nous récupérons la taille courante de l'ecore_evas et l'utilisons pour redimensionner l'objet trans à la taille de la fenêtre. Une fois ceci fait nous affichons le trans et le ecore_evas. Dernièrement, nous raffraichissons l'image de transparence en fonction de ce qui est présent dessous.

Exemple 6.9. makefile simple

```
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)
```

Pour compiler le programme précédent nous devons inclure les informations de librairies pour ecore, ecore_evas et esmart. Cela est fait par l'utilisation des scripts -config pour chaque librairie. Ces scripts -config savent où chaque librairie réside et configurent les chemins d'inclusion et de liens appropriés pour la compilation.

Recette: Introduction au containeur Esmart

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

Il est souvent nécessaire, lors de la création de l'interface d'un programme de grouper certains éléments et que leur mise en page dépende d'un autre. Dans ce but, la librairie containeur Esmart à été créée. Elle à été développée dans le but de prendre en charge la partie de la mise en page, et dans le cas où elle ne comble pas vos besoins, les partions de mise en page du containeur sont extensibles et changeables.

Cette recette fournira les base de l'utilisation d'un containeur Esmart. Le produit final sera un programme qui vous permettra de voir certaines des différentes combinaisons de mises en page du containeur par défaut. La mise en page sera assurée par Edje avec des callbacks sur le programme pour modifier la mise en page du containeur et indiquer si l'utilisateur à cliqué sur un élément containeur.

Exemple 6.10. Déclarations et fichiers En-Tête

```
#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"};
```

Comme les autres programmes EFL nous devons inclure Ecore, Ecore_Evas, Edje et puisqu'il s'agit d'un exemple de containeur l'en tête Esmart/Esmart_Container. Getopt sera utilisé pour le traitement de la ligne de commande.

Puis viennent des prototypes de fonctions qui serons décrites plus tard lorsque que nous nous intéresserons à leur implémentation. Puis quelques variables globales que nous utiliserons à travers le programme. Le tableau str_list est le contenu qui sera stocké dans le containeur.

Exemple 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);
}
```

Le début de la fonction main récupère les options de la ligne de commande et configure l'affichage par défaut. Comme vous pouvez le voir, nous avons besoin d'un thème à afficher. Cela peut être fait plus intelligement, en cherchant dans les répertoires d'installation par défaut et dans le dossier application de l'utilisateur, mais cet exemple va au plus simple et force le thème à être indiqué en ligne de commande.

Exemple 6.12. Initialization

```
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();
```

Après avoir reçu les arguments de la ligne de commande, nous procédons à l'initialisation des librairies requises, Ecore, Ecore_Evas et Edje. Nous configurons aussi le taux de refraichissement Edje.

Une fois l'inintialisation terminée nous créons l'interface principale de l'application. J'ai séparé la création du containeur dans une fonction à part pour rendre le code du containeur plus simple à localiser dans l'exemple.

Une fois que tout est créé nous appelons `ecore_main_loop_begin` et attendons qu'un évennement se produise.

Exemple 6.13. Shutdown

```
edje_shutdown();
```

```
EXIT_ECORE_EVAS:
    ecore_evas_shutdown();

EXIT_ECORE:
    ecore_shutdown();

EXIT:
    return ret;
}
```

Le procédure de fin usuelle, soyons de bons programmeurs et fermons tout ce que nous avons démarré.

Exemple 6.14. callbacks de fenêtre

```
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);
        }
    }
}
```

Ensuite, nous configurons quelques callbacks génériques qui seronts utilisés par l'interface. Ceux ci seront les callbacks exit, destroy et resize. Une fois de plus, les fonctions de style EFL usuelles. Cependant le callack exit n'est pas strictement nécessaire car Ecore lui même appellera `ecore_main_loop_quit()` si aucun handler n'est enregistré pour ce callback.

Exemple 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);
}
```

L'interface consiste en l'objet Ecore_Evas qui contient le canvas lui même, et l'objet Edje que l'on utilisera pour contrôler notre mise en page. La fonction `make_gui` configure les callbacks définis ci-dessus et créé l'objet Ecore_Evas.

Une fois que nous avons définit l'objet Evas et les callbacks, nous créeons l'objet Edje qui définira notre mise en page. L'appel à `edje_object_add` est utilisée pour créer l'objet sur l'objet Evas, et une fois que cela est fait, nous prenons le thème indiqué par l'utilisateur et configurons notre objet Edje pour qu'il utilise ce thème, le paramètre "container_ex" est le nom du groupe que nous utilisons dans l'objet EET.

Une fois que le fichier thème est configuré sur l'objet Edje, nous utilisons les valeurs dans le fichier thème pour configurer les pages de taille pour notre application, et affichons l'objet Edje. Nous configurons ensuite les callbacks pour les objets Edje et Ecore_Evas.

Exemple 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);
}
```

Le programme aura deux callbacks principaux attachés à l'objet Edje, l'un pour le click avec le bouton gauche de la souris et l'autre pour le bouton droit. Ils seront utilisés pour modifier la direction et l'alignement du containeur. Les deuxième et troisième paramètres des callbacks doivent vérifier la donnée emmise par Edje, cela sera vu plus tard lorsque nous nous intéresserons au fichier EDC. Le troisième paramètre est la fonction à appeler et le dernier, n'importe quelle donnée que nous souhaiterions passée au callback.

Exemple 6.17. container_build

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

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

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

La fonction `container_build` créera le containeur et configurera nous éléments de données dans le containeur cité. La création est assez simple étant donné que l'appel à `esmart_container_new` renvoi un objet Evas_Object qui est le containeur lui même. Une fois le containeur créé, nous pouvons lui configurer un nom pour que les prochaines références à celui-ci soient plus simples.

Ensuite nous configurons la direction, qui peut être (CONTAINER_DIRECTION_VERTICAL ou CONTAINER_DIRECTION_HORIZONTAL) [ou dans ce cas, un "int" passé depuis la ligne de commande lié à 1 ou 0 respectivement]. La direction indique au containeur dans quelle direction les éléments seront déssinés.

Après la direction nous configurons l'alignement de notre containeur. L'alignement indique au containeur où déssiner les éléments. Les valeurs possibles sont CONTAINER_ALIGN_CENTER, CONTAINER_ALIGN_LEFT, CONTAINER_ALIGN_RIGHT, CONTAINER_ALIGN_TOP et CONTAINER_ALIGN_BOTTOM. Avec la valeur par défaut, gauche et droite ne s'appliquent qu'a un containeur vertical, et le haut et bas ne s'appliquent qu'a un containeur horizontal, évidemment centré s'applique aux deux.

Si nous désirons utiliser une mise en page différente que celle par défaut, nous pouvons faire un appel à `esmart_container_layout_plugin_set(containeur, "nom")` où le nom est le nom du plugin que nous souhaitons utiliser. Le paramétrage par défaut est le containeur appelé "default".

Une fois les directions et alignements configurés, l'espacement et les marges du containeur sont spécifiés. Les marges spécifient les espaces autour de l'extérieur du containeur et requièrement quatres paramètres numérique : gauche, droite, haut et bas. L'espacement spécifie l'espacement entre les différents éléments du containeur.

Nous continuons donc avec la police de remplissage du containeur. Cela indique comment les éléments sont positionnés pour remplir l'espace disponnible dans le containeur. Les valeurs possibles sont : CONTAINER_FILL_POLICY_NONE, CONTAINER_FILL_POLICY_KEEP_ASPECT, CONTAINER_FILL_POLICY_FILL_X, CONTAINER_FILL_POLICY_FILL_Y, CONTAINER_FILL_POLICY_FILL et CONTAINER_FILL_POLICY_HOMOGENOUS.

Uns fois que le containeur est totalement configuré nous définissons les couches de celui-ci, puis appelons la fonction "swallow" afin que les redimensions s'appliquent à notre containeur.

Exemple 6.18. Ajouter des éléments au Containeur

```
    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);
}
```

Maintenant que nous avons notre containeur, nous pouvons y ajouter quelques éléments pour les afficher. Chacune des entrées définie dans le tableau str_list au début du programe sera ajoutée dans le containeur comme composante texte.

Pour chaque élément nous créeons un nouvel objet Edje sur notre objet Evas. Nous devons ensuite connaitre le nom du fichier de thème utilisé pour créé notre Edje principal, sont nous appelons la fonction `edje_object_file_get` qui configurera le fichier Edje avec la valeur donnée.

Nous essayons ensuite de configurer le groupe nommé "element" sur l'élement nouvelement créé. Si cela ne fonctionne pas nous affichons une erreur et supprimons l'objet.

Si nous avons trouvé le groupe nommmé "element" nous pouvons essayer d'obtenir la partie "element.value" pour notre élément. Si cette partie existe, nous configurons la valeur texte comme partie de notre chaîne courante et la montrons.

Un callback est créé au travers de `edje_object_signal_callback_add` et attaché au nouvel élément. Cela sera appelé si le signal "item_selected" est envoyé par Edje. La valeur i_ptr montre comment les données peuvent être attachées à l'élément. Lorsque l'utilisateur cloque sur un élément, sont nombre est affiché sur la console.

Uns fois l'élément créé nous l'ajoutons au containeur (dans ce cas, concatainer l'élément).

Pour finir, le containeur est affiché et nous réalisons quelques taches supplémentaires pour afficher des informations relatives au cotnainer dans l'en-tête au travers de l'appel à `_show_text`.

Exemple 6.19. _set_text

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

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

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

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

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

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

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

La routine `_set_text` prend la direction et l'alignement courants du containeur et configure quelques textes dans l'en-tête du programme. Il s'agit juste d'un communication simple avec l'utilisateur pour l'informer des configuration courantes du containeur.

Exemple 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);
}
```

Lorsque l'utilisateur clique avec le bouton gauche de la souris sur l'ecran ce callback est exécuté. Nous prenons l'information courante sur la direction du containeur et allons vers une autre directions (horizontal devient vertical, etc...). Nous réinitialisons aussi l'alignement si nous sommes pas alignés au centre pour être sûrs que tout est validé pour la nouvelle direction. Le texte de l'en-tête est mis à jour avec les nouvelles configurations.

Exemple 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));
}
```

Le callback du click droit va boucler sur les alignements disponnibles pour une direction donnée lorsque l'utilisateur cliquera sur le bouton droit de la souris.

Exemple 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));
}
```

Enfin le callback `_item_selected` sera exécuté lorsque l'utilisateur cliquera avec le bouton du milieu sur un élément du containeur. La donnée contiendra le nombre configuré pour cet élément dans la routine de création précédente.

Ceci est la fin du code pour notre application, viens maintenant le code requis pour l'EDC afin que tout s'affiche correctement.

Exemple 6.23. L'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;
                }
            }
        }
```

Ce fichier EDC essaye d'inclure la police de caractère Vera, comme définit dans la section police au début. Cela signifie que lorsque vous compilerez edc vous aurez besoin de la police de caractère Vera dans le dossier ou donner à edje_cc le paramètre -fd et spécifier le dossier des polices de caractères.

Une fois que les polices de caractères sont définis les collections principales sont définies. La première collection est la première portion de l'application elle même, le groupe "container_ex". Ce groupe spécifie la fenêtre principale de l'application. En tant que telle elle comporte le fond, l'en-tête et le texte d'en-tête. Ces parties sont quasiments standard avec quelques alignements (minimums) réalisés entre eux.

Exemple 6.24. La partie Containeur

```
            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";
        }
    }
}
```

Le partie containeur est ensuite définie. La partie elle même est assez simple, simplement positionnée relativement au fond et configurée pour recevoir les évènements de la souris. Une fois les parties définies nous spécifions les programmes pour ce groupe, qui sont au nombre de deux. Le premier programme "left_click" spécifie ce qui va arriver lorsqu'un click est réalisé sur le bouton gauche de la souris.

L'action est d'emmêtre un signal, les deux paramètres après SIGNAL_EMIT correspondent aux valeurs données dans le callback du code de l'application.

Il existe un callback similaire pour le bouton droit de la souris identique à celui du premier, déclanchant simplement un signal différent.

Exemple 6.25. Le groupe Element

```
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";
        }
    }
```

```
    }
}
```

Le groupe élément spécifie comment chaque élément du containeur doit être affiché. Vous noterez que les noms donnés correspondent les noms recherchés dans le code de l'application elle même lors de la création des éléments.

Il y a un programme dans ce groupe qui émettra un signal "item_selected" lorsque le bouton du milieu de la souris sera utilisé au dessus d'un des éléments de la liste.

Ceci marque la fin du code de l'EDC. Pour compiler le code de l'application, un makefile similaire au suivant peut être utilisé.

Exemple 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)
```

t pour créer e fichier EET, un simple 'edje_cc default.edc' devrait suffire si le fichier de la police de caractères Vera.ttf est dans le dossier courant.

Voila, en supposant que tout s'est passé comme prévu, vous devriez disposer d'une application simple dans laquelle cliquer sur les boutons gauche et droit de la souris déplace le containeur dans différentes portions de la fenêtre. Lorsque vous cliquez sur le bouton du milieu sur un élément, le numéro de celui-ci est affiché.

Chapitre 7. Epeg & Epsilon

Dans notre ère moderne de la photographie numérique, l'affichage devient un problème face au volume énorme des images qui sont crées. A la différence de l'ancien temps, lorsque le film était employé avec parcimonie, les images sont maintenant produites hebdomadairement par centaines ou milliers. La solution à ce problème de présentation est d'utiliser des imagettes (ou vignettes), des images réduites qui peuvent être classées dans une table ou une application et être rapidement balayées visuellement pour trouver les images que vous désirez. Mais le redimensionnement d'image est une opération fastidieuse. Même si cela pourrait ne prendre qu'une seconde à votre puissant AMD Athlon pour réduire une photo de 1600x1200 à la résolution désirée, si vous avez 2000 photos à réduire cela vous prendra 30 minutes, en partant du fait que vous ne faites pas l'opération à la main dans un éditeur tel que Photoshop ou GIMP. Le problème réclame clairement un outil qui peut réduire des images avec vitesse et efficacité, et autant de paramètres que possible. On peut répondre à ce problème grâce à deux bibliothèques faisant partie d'EFL: Epeg et Epsilon.

Epeg a été écrit par Raster afin de traiter le problème mentionné ci-dessus qu'il avait rencontré avec les galeries d'images de son site rasterman.com. C'est, sérieusement, le créateur de vignettes le plus rapide de la planète. Avec une API facile à utiliser, il peut être intégré à n'importe quelle application. Son seul défaut est qu'il ne sait manipuler que les JPEGs (d'où son nom), mais c'est un problème minime si l'on considère que tous les appareils-photo du marché utilisent le JPEG comme format de sortie par défaut.

Epsilon a été écrit par Atmos, en s'inspirant de la vitesse fulgurante d'Epeg mais en réponse à un besoin de réduction multi-format. Epsilon peut manipuler le JPEG, le PNG, le XCF, et le GIF. Evidement, puisque ce n'est pas une bibliothèque spécialisée dans le JPEG, il ne le manipule pas aussi vite qu'Epeg, mais il peut employer Epeg lui-même afin de gagner en vitesse. Epsilon, contrairement à Epeg, se conforme à la norme de gestion de vignette de freedesktop.org Thumbnail Managing Standard [http://triq.net/~jens/thumbnail-spec/index.html]. En tant que tel, il produit toutes les imagettes dans le répertoire spécifié par le standard (~/.thumbnails/) plutôt qu'à un endroit définit par le programmeur.

Ces deux bibliothèques accomplissent des tâches si spécifiques que leurs APIs sont très simple à utiliser. Epeg a seulement 17 fonctions et Epsilon seulement 9, ce qui les rend faciles à apprendre et à utiliser tout de suite dans vos applications.

Recette: Miniaturisation d'Image avec Epeg

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

L'application de réduction d'image la plus simple que l'on puisse écrire prendrait seulement deux arguments, le nom de fichier d'entrée (l'image) et le nom de fichier de sortie (la vignette). L'exemple de code qui suit utilise Epeg pour faire cela.

Exemple 7.1. Vignette 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);
}
```

Cet exemple est assez simpliste, on ne vérifie pas que l'entrée est vraiment un JPEG, mais on utilise de façon adéquate quelques dispositifs de la bibliothèque. Il peut être compilé de la façon suivante:

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

La fonction `epeg_file_open` ouvre le JPEG à manipuler, retournant un pointeur Epeg_Image. Ce pointeur peut être fourni à d'autres fonctions Epeg pour la manipulation.

Deux différentes fonctions sont utilisées ici pour recueillir quelques informations sur l'image d'entrée: `epeg_size_get` et `epeg_comment_get`. Noter qu'aucune des valeurs de retour de ces fonctions ne sont utilisées dans d'autres fonctions Epeg, elle servent seulement à l'affichage d'informations. Une bonne utilisation de ces valeurs pourrait être de définir intelligement la taille de l'imagette de sortie ou de modifier et transmettre un commentaire à cette image.

L'ensemble suivant de fonctions effectuent le réel travail. `epeg_decode_size_set` définit la taille de l'imagette. `epeg_file_output_set` définit le nom de fichier de sortie. Et `epeg_encode` s'occupe du lifting. Notez que bien que nous ne vérifions pas le succès de cette opération, `epeg_encode` retourne un int qui nous le permettrait.

Une fois que la vignette est créée, il suffit d'appeler `epeg_close` pour sceller l'affaire.

Bien que cet exemple soit peut-être extrèmement simpliste, il vous permet de voir fonctionnement de base d'Epeg. Ce dernier possède également des fonctions pour le découpage, les commentaires, pour activer ou désactiver les commentaires des vignettes, la conversion de couleur et les changements de paramètres de qualité, qui peuvent être utilisées pour obtenir exactement le résultat que vous voulez.

Recette: Miniaturisation d'Image avec Epsilon

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

Epsilon crée des vignettes conformes au Thumbnail Managing Standard [http://triq.net/~jens/thumbnail-spec/index.html] de freedesktop.org. Les vignettes peuvent être créées depuis une grande variété de formats, incluant le support natif PNG, le support Epeg, ou n'importe quel format supporté par Imlib2. Jettons un oeil à une application Epsilon similaire à l'exemple Epeg vu plus tôt.

Exemple 7.2. Vignette 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);
}
```

Ceci peut être compilé de la manière suivante:

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

Vous noterez presque immédiatement qu'aucun nom de fichier de sortie n'est accepté, de même qu'aucune fonction de sortie n'est utilisée. La norme de gestion de vignettes freedesktop.org indique que toutes les imagettes doivent être créées dans l'arborescence ~/.thumbnails. Ce stockage centralisé des vignettes tiens compte du partage de ces dernières entre les applications qui adhèrent à la norme. Après avoir compilé et lancé l'exemple de code avec une image, cherchez l'imagette dans ~/.thumbnails/large. Les imagettes sont également nommées selon le standard, en remplaçant le nom original par une somme MD5 de sorte que même si l'image de base est renommée, la vignette n'a pas besoin d'être régénéré.

Dans notre exemple nous commençons par vérifier que nous obtenons bien une image d'entrée à miniaturiser et initialisons alors Epsilon grâce à la fonction `epsilon_init`. `epsilon_new` accepte un seul argument, l'image à réduire, et renvoie un pointeur epsilon qui sera employé par les autres fonctions.

Epsilon a la capacité de tirer des informations basiques de vos images. Dans l'exemple ci-dessus nous employons `epsilon_info_get` pour renvoyer une structure Epsilon_Info contenant la date de modification de l'image d'entrée (mtime), le lieu (URI), la largeur, la hauteur et le type MIME. Ici nous rapportons simplement la largeur et la hauteur de l'image en utilisant les éléments `w` et `h` de la structure informative.

`epsilon_generate` fait le gros du travail. Cette fonction produira la vignette et la placera à l'endroit approprié. Sa valeur de retour indique s'il y a eu succès, et l'en-tête Epsilon fournit des définitions de macro CPP: `EPSILON_FAIL` et `EPSILON_OK`.

Le nettoyage est fait par `epsilon_free`.

Comme nous l'avons vu ici, Epsilon est très simple à utiliser et à intégrer dans n'importe quelle application en rapport avec la miniturisation d'image. Non seulement il apporte une API simple, mais il intègre une norme régnante sans frais supplémentaires. Pour plus d'informations sur Epsilon, voyez le Doxygen Epsilon sur Enlightenment.org.

Chapitre 8. Edje

Edje est une bibliothèque complexe de design graphique. Son but est de soustraire chaque éléments de l'interface de votre application Evas du code proprement dit.

Une application Edje se divise en deux parties: le code C qui constitue votre application et une Collection de Données Edje (EDC) qui décrit tous les éléments de votre interface. Ces deux parties sont connectées grâce à des signaux émis par votre EDC et reçus par des callbacks dans le code de votre application. L'utilisation de ce modèle de signaux fait que le code se désintéresse complètement de ce à quoi l'interface ressemble tant qu'il reçoit un signal. Et comme les signaux sont pris en charge par des callbacks, votre interface n'est pas obligée d'envoyer tous les signaux possibles, permettant ainsi de créer de grosses applications et des plus petites avec un seul binaire. Que votre interface utilise des boutons ou une drag-barre pour envoyer des données, votre application ne s'en soucie pas.

Recette: Un modèle de construction d'applications Edje

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

L'exemple suivant est un modèle qui peut être utilisé pour la mise en oeuvre rapide d'une application Edje. Il est assez similaire au modèle que l'on peut trouver au chapitre Evas, dans la mesure où il utilise aussi Ecore_Evas.

Exemple 8.1. Modèle 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;
}
```

Compilez ce modèle de la façon suivante:

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

Les points importants sont contenus dans le bloc Edje, suivant `edje_init()`.

`edje_object_file_set()` définit quel Edje EET est utilisé et également le nom de la collection à employer.

Le reste des fonctions Edje/Evas du bloc Edje sont nécessaires pour redimensionner la fenêtre X11 afin de l'accommoder à votre Edje. Nous commençons par déplacer la fenêtre Evas puis par obtenir la taille minimum de l'Edje en lui-même avec `edje_object_size_min_get()`. Ensuite, grâce à `evas_object_resize()` nous pouvons redimensionner l'Edje, qui est en réalité un objet Evas, à la taille de l'Evas en lui-même. Après cela nous affichons l'Edje et redimensionnons l'Evas (et grâce à Ecore, la fenêtre aussi) avec `ecore_evas_resize()`.

Au delà de ceci, des callbacks peuvent être ajoutés et liés à votre interface.

Recette: Création/Déclenchement de callbacks Edje

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

Il est parfois nécessaire de signaler à votre programme principal qu'un événement s'est produit au niveau de l'interface utilisateur, mais on ne veut généralement pas que l'implémentation bave sur le design de l'UI. Avec Edje on peut régler ce problème en déclenchant un signal depuis un programme EDC et en attachant un callback à ce signal dans le programme en C.

Exemple 8.2. Programme 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);
}
```

La plupart de ce qu'on voit ici est une mise en place standard d'Ecore, Ecore_Evas et Edje. Le callback est attaché avec `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)`. L'objet o auquel le callback est attaché est l'objet Edje qui a été créé avec votre fichier EDC.

Les valeurs `emission` et `source` doivent être des chaînes qui correspondent aux appels d'émission du programme EDC qui sera vu plus tard. L'autre option est de placer un '*' dans `emission` ou `source`. Ainsi, la valeur correspondra à n'importe quel signal. Si vous voulez intercepter tous les signaux que edje emet, vous pouvez placer l'emission et la source à '*'.

`func` est la fonction à appeler et, pour finir, `user_data` correspond à n'importe quelles données supplémentaires que vous voudriez passer au callback.

Vous pouvez apercevoir la fonction de callback `edje_cb`. Elle recevra les données utilisateur, l'objet Edje dont provient le callback et les chaînes emission et source.

Pour activer le callback votre fichier EDC a besoin d'un programme qui émettra l'emission et la source requise.

Exemple 8.3. Fichier 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";
            }
        }
```

```
    }
}
```

La portion intéressante est `action: SIGNAL_EMIT "foo" "bar"` qui causera de la part d'Edje l'émission de `foo` depuis la source `bar`.

Exemple 8.4. Compilation

```
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 rend très simple la séparation de l'interface de son implémentation. L'interface doit seulement pouvoir envoyer les émissions et sources au moment où arrivent les événements.

Recette: Travailler avec des fichiers Edje

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

Lorsqu'on travaille avec des fichiers .edc et .eet on a souvent besoin de transformer l'un en l'autre. Pour faire ça Edje fourni un ensemble d'outils pour faciliter les transformations.

Les programmes disponibles sont:

edje_cc	Compile un fichier EDC, les images et les fontes en un fichier EET
edje_decc	De-compile un fichier EET en un fichier EDC, des images et des fontes
edje_recc	Re-compile un fichier EET
edje_ls	Liste les groups d'un fichier EET
edje	Affiche les groupes d'un fichier EET

Chacun de ces programmes sont expliqués plus en détail ci-dessous.

edje_cc

`edje_cc` est un des principaux programmes Edje que vous utiliserez. Il est responsable de la compilation de vos fichier EDC, incluant images et fontes en un fichier EET correspondant.

Exemple 8.5. Utilisation de edje_cc

```
edje_cc [OPTIONS] input_file.edc [output_file.eet]
```

Options

-id *image/répertoire*	Ajoute un répertoire pour la recherche des images
-fd *font/directory*	Ajoute un répertoire pour la recherche des fontes
-v	Affichage verbeux
-no-lossy	Ne pas autoriser la dégradation des images
-no-comp	Ne pas autoriser le stockage des images avec une compression sans perte
-no-raw	Ne pas autoriser le stockage des images sans compression (raw)
-min-quality *VAL*	Ne pas autoriser les images dégradées avec une qualité < VAL (0-100)
-max-quality *VAL*	Ne pas autoriser les images dégradées avec une qualité > VAL (0-100)
-scale-lossy *VAL*	Redimensionne les pixels des images dégradées par ce facteur de pourcentage (0 - 100)
-scale-comp *VAL*	Redimensionne les pixels des images compressées sans perte par ce facteur de pourcentage (0 - 100)
-scale-raw *VAL*	Redimensionne les pixels des images non compressées (raw) par ce facteur de pourcentage (0 - 100)
-D *define_val=to*	Define dans le style CPP pour définir les définitions d'entrée macro de la source .edc

edje_decc

`edje_decc` permet de décompiler les fichiers EET, redonnant ainsi les EDC, les images et les fontes. Cela facilite la distribution de vos sources puisque vous n'avez besoin de fournir que le fichier EET et l'utilisateur final aura accès aux sources et au produit finit.

Exemple 8.6. Utilisation de edje_decc

```
edje_decc input_file.eet
```

edje_recc

`edje_recc` permet de recompiler un fichier EET sans avoir à d'abord le décompiler. Cela permet de modifier les paramètres passés à `edje_cc` pour mieux s'accorder à vos besoins esthétiques et à la taille voulue de l'EET.

Exemple 8.7. Utilisation de edje_recc

```
edje_recc [OPTIONS] input_file.eet
```

Options

-v	Affichage verbeux
-no-lossy	Ne pas autoriser la dégradation des images
-no-comp	Ne pas autoriser le stockage des images avec une compression sans perte
-no-raw	Ne pas autoriser le stockage des images sans compression (raw)
-min-quality *VAL*	Ne pas autoriser les images dégradées avec une qualité < VAL (0-100)
-max-quality *VAL*	Ne pas autoriser les images dégradées avec une qualité > VAL (0-100)

edje_ls

`edje_ls` fournit une liste de tous les groupes d'un fichier EET donné. C'est une façon rapide de voir ce que renferme un EET.

Exemple 8.8. Utilisation de edje_ls

```
edje_ls [OPTIONS] input_file.eet ...
```

Options

-o *outputfile.txt*	Ecrit la liste des collections dans un fichier

edje

`edje` est aussi un des principaux programmes que vous serez amené à utiliser. `edje` permet de voir chacun des groupes de votre programme, ce à quoi les différentes parties vont ressembler et comment elles réagissent à certain signaux.

Exemple 8.9. Utilisation de edje

```
edje file_to_show.eet [OPTIONS] [collection_to_show] ...
```

Options

-gl	Utiliser OpenGL pour le rendu
-g *W*x*H*	Règle la taille de la fenêtre à WxH
-fill	Fait en sorte que les parties remplissent toute la fenêtre

Ces cinq outils devraient fournir tout ce dont vous avez besoin pour contruire et maintenir vos EETs. Il rendent aussi facile le rappartiement de la source comprise dans un EET, aidant ainsi à en apprendre le fonctionnement.

Chapitre 9. Edje EDC & Embryo

Les fichiers sources de la Collection de données Edje (Edje Data Collection EDC) autorisent la création simple d'interface graphiques riches et puissantes. Votre application Edje est divisée en deux parties distinctes, la partie code source (utilisant des appels de `Edje.h`) et la description de l'interfac dans l'EDC. La seule connexion nécessaire entre votre interface et le code de votre application sont les signaux émis par votre interface et recus par les callbacks Edje dans votre code.

Un EDC est divisé entre plusieurs sections majeures décrivant les images et les polices de caractères utilisées par votre interface, la descriptions de comment les différentes `parties` de votre interfaces sont mises en page, et les descriptions de comment votre `programme` réagit lorsque l'on agit avec l'interface. Cete fonctionalité peut être offerte par l'emploie du langage de script d'Embryo pour ajouter une programmation ressemblant à C dans EDC lui même.

Le résultat d'un EDC, incluant toutes ses images et ses polices de caractères, est un unique EET. Etant donné que l'interface complète résulte en un simple fichier, la distribution de thèmes est très simplifiée.

Tandis que les EDCs de Edjes peuvent être considérés comme des thèmes, ils peuvent faire bien plus. Un thème traditionnnel est un fichier ou un groupe de fichiers améliorant une interface existante en modifiant la couleur des éléments et en remplacant les images qui font l'interface elle même. Mais ces méthodes sont insuffisantes pour changer réellement le design de l'interface d'une application, limitent les créateurs de thèmes et requièrent souvent un remaniement de l'interface pour plus de fonctionalités. Une application GTK ressemblera toujours à la même chose quelque soit le thème utilisé. Un exemple simple serait qu'une application GTK ou QT aurais toujours une forme rectangulaire et si elle avait une bordure, vous ne pourriez pas l'enlever à l'aide d'un thème. Cependant, une application Edje peut passer d'une forme extérieure rectangulaire à ovale grâce à une simple modification de l'EDC., ou vous pourriez supprimer ou réarranger tous les éléments de l'interface sans même toucher le code source. Dans ce sens, Edje autorise bien plus de controle et de flexibilité que n'importe quelle autre solution de la communauté Open Source et autorise un modèle de programmation Ouvert et autorise même des personnes ne sachant pas programmer (comme la plupart des créateurs de thèmes) à apporter leur contribution et modifier les choses comme ils les voient.

Recette : "Toggle" Edje/Embryo

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

Il y à longtemps, Raster [http://www.rasterman.com] créeat Edje, et cela était bon. Les hommes vivant dans les murs de la cave (#edevelop) furent impressionnés, mais avant cela il y eu plusieurs essais. Cela donna beaucoup de créativité mais il fallait recourrir à l'alchimie pour que tout se passe bien. Pour des raisons historiques, un "Toggle" Edje sans embryo à été conservé. Voyez l'exemple Edje sans Embryo ci dessous.

Vous noterez que vous devez parler en signaux à votre application pour déterminer l'état de votre "toggle". Donc, sans autre additifs, voici un "toggle" Edje utilisant embryo, avec une méthode *plus* élégante.

Le scripting Embryo dans Edje, désormais scripting EE, vous procure des variables. Vous pouvez avoir des entiers, des nombres a virgule et des chaines. Cela signifie que vous pouvez disposer basiquement d'une logique programmation dans vos edjes. Rien de compliqué comme les structures, mais des variables simples contenus dans des groupes peuvent ressembler à des structures.

La première partie de EE est de choisir vos variables. Dans cet example simple nous n'avons qu'une seule variable, et nous nous incluons dans un group edje en déclarant un bloc *script { ... }*. *button_toggle_state* est implicitement un entier, et sera utilisé comme valeur booléenes pour que nous sachions si notre bouton "toggle" est enclenché ou pas. Ce qui est intéressant avec cette variable c'est que nous pouvons l'utiliser comme moyen de communication entre notre application et notre edje. Désormais vous pouvez savoir facilement (si vous l'avez correctement réalisée) si une action edje va envoyer votre application dans l'oubli.

Exemple 9.1. Creation des variables

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

La seconde partie du scripting EE est l'initialisation de vos ariables. Ces variables sont initialisées à zéro, mais c'est un bon exercice de les initialiser vous même. Edje émet un signal "load" quand le groupe est chargé dans la mémoire, c'est une oportunité pour configurer vos variables.

Exemple 9.2. Initialisation des variables

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

La troisième partie est de donner un look avec votre edje. Pour cet exemple des rectangles sont utilisés mais des images et des textes devraient aussi fonctionner correctement. Il y a un objet "fond" pour un peu plus de consistance et un rectangle appelé "toggler". toggler a deux états, celui par défaut (implictement désactivé) et activé. Lors que l'on clique sur toggler il doit, vous l'imaginez bien, changer d'etat. Déactivé -> activé, activé -> désactivé. Toggler aura son état par défaut (désactivé) rouge, et son état activé bleu pour que l'on puisse facilement les différencier. Le fond sera blanc car ce n'est ni bleu ni rouge :D

Exemple 9.3. Le bouton 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);
                }
            }
        }
    }
}
```

La quatrième partie est la capture des évènement des la souris pour les pogramme edje. Pas seulement changer la variable Embryo, mais aussi changer l'apparence de notre edje. Cet exemple utilise le click gauche normal de la souris, en termes edje, "mouse,clicked,1". Cet exemple n'utilise la fontion Embryo *set_state* mais émet des signaux capturés par les autres programmes. Le raisonnement développé derrière cela est de permettre des transitions visuelles entre les deux états. La fonction set_state d'Embryo est un changement d'état immédiat, et n'est pas aussi beau que la transitions SINUSOIDAL utilsée dans le morceau de code suivant.

Exemple 9.4. Capturer les évennement de la souris

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

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

La cinquième partie est de poser le scénario présenté. Cela n'est que la partie émergée de l'iceberg en ce qui concerne le scripting EE. Vous pouvez ajouter bien plus de variables pour conserver la traces d'états internes ne concernant pas du tout votre application. Il y a des nuances entre cet utilisation et celle l'utilisation pratique des variables Embryo, mais comprendre ces blocs rendra le travail avec des applications utilisant le scripting EE plus simple.

- Qui a-t'il de mauvais dans la technique presentée ici ?
- Comment faire si l'application à besoin d'un toggle "activé" par défaut ?

Vous pouvez utiliser un script similaire à celui-ci pour construire vos applications.

Exemple 9.5. Script de construction

```
#!/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
```

Exemple 9.6. Toggle Edje sans 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;
     }
        }
    }
}
```

Recette : Fondu de texte avec Edje

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

Les effets de texte peuvent donner une très belle apparence à votre application. Mais comment faire si vous

souhaitez fondre ces effets avec votre texte ? Bien Edje rend cela possible et relativement simple.

Tout ce que vous avez besoin de faire est de fondre l'attribut `color3` de votre texte à `color`. La `color3` changera les valeur de couleurs de l'effet.

Cela est illustré dans l'exemple suivant.

Exemple 9.7. Effet de fondu avec du texte

```
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, "mouse.in";
                signal, "mouse,in";
                source, "foo";
```

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

Ces exemples peuvent être compilés dans un `.eet` avec la commande suivante.

Exemple 9.8. Compilation

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

En modifiant la valeur `color3` et a valeur `color` vous pourrez modifier l'apparence des effets de votre texte.

Chapitre 10. EWL

La "Enlightened Widget Library" (EWL) est une boite à outils de "wigdets" (gadgets de fenêtre) qui est construite au dessus des fondation créés par les autres librairies de EFL. Ewl utilise Evas pour son affichage, et son apparence est gérée par Edje.

Ewl est similaire dans son fondement, à plusieurs autres boites à outils, dont GTK, QT ou MOTIF. L'API diffère mais les concepts sont les mêmes.

Recette : Introduction à EWL

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

Au travers de l'utilise de Enlightened Widget Library (EWL), beaucoups de puissance peut être mise dans les mains du programmeur sans que celui-ci ai à faire beaucoup d'efforts.

Cette introduction à EWL vous montrera comment créér une simple application de visionnement de texte avec une barre de menu et une fenêtre de gestion de fichiers. La zone de texte disposera d'ascenseurs et autorisera le défillement aussi bien à l'aide du clavier que de la molette de la souris.

Exemple 10.1. Inclusions et déclarations

```
#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);
```

La seule inclusion nécessaire pour la réalisation d'une application EWL est la déclaration d'<Ewl.h>. Nous créeons la fenêtre principale et la fenêtre de dialogue globale pour rendre plus facile l'accès aux fonctions de callback. Ils n'ont pas besoin d'êtres globaux mais pour l'exemple il est plus simple qu'ils le soient.

Exemple 10.2. main

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

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

La fonction main pour notre visionneur de texte est très simple. Nous commencons par initialiser ewl par l'appel à `ewl_init()`. Ewl prend les paramètres argc et argv pour faire quelques traitements de la ligne de commande par elle même. Cela inclu certaines choses comme régler le thème Ewl à utiliser (--ewl-theme) ou configurer le moteur de rendu à utiliser (--ewl-software-x11, --ewl-gl-x11, etc.).

`ewl_init()` fait avec attention le travail d'initialisation des autres librairies, rendant invisible ce travail au développeur et lui fournissant une interface simple.

L'appel à mk_gui configurera la fenêtre principale et tous les contenus requis.

L'appel à `ewl_main()` configure la boucle principale de traitement, et jusqu'a la sortie la prise en charge de toutes les applications devant être quittés, de ce fait il n'y as pas d'appel à "shutdown" dans notre routine main.

Exemple 10.3. mk_gui : création de la fenêtre

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

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

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

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

La première chose que nous devons faire pour faire naitre notre application est de créér la fenêtre principale. Cela est fait grace à l'appel à `ewl_window_new()`. Une fois que nosu avons la fenêtre nous pouvons continuer par la configuration du titre (comme il apparaîtra dans la barre du gestionnaire de fenêtre au dessus de l'application), le nom et la classe de la fenêtre.

Une fois les informations traditionelles configurées pour la fenêtre la taille par défaut de celle-ci est configuré à 200x300 grace à l'appel à `ewl_object_size_request()`. En même temps que la taille par défaut nous pourrions configurer la taille minimum et maximum de la fenêtre avec des appels à `ewl_object_minimum_size_set()` et `ewl_object_maximum_size_set()`. Mais ceci n'est pas nécessaire par notre application donc nous ne le faisons pas.

Le configuration finale de l'application est faite par la configuration de la police de remplissage avec `ewl_object_fill_policy_set()`. Cela configure comment EWL va grouper les "widget" dans la fenêtre, aves les valeurs possibles :

EWL_FLAG_FILL_NONE	Ne pas étirer ou rétrécir dans quelque direction que ce soit

EWL_FLAG_FILL_HSHRINK	Rétrécir horizontalement
EWL_FLAG_FILL_VSHRINK	Rétrécir verticallement
EWL_FLAG_FILL_SHRINK	Rétrécir horizontalement et verticalement
EWL_FLAG_FILL_HFILL	Remplir horizontalement
EWL_FLAG_FILL_VFILL	Remplir verticalement
EWL_FLAG_FILL_FILL	Remplir horizontalement et verticalement
EWL_FLAG_FILL_ALL	Rétrécir et remplir en même temps

Une fois que toutes les propriétés de la fenêtre sont définies un callback pour capter pour la destruction de la fenêtre est attaché par `ewl_callback_append()`. La fonction `destroy_cb()` sera appelée si quelqu'un demande la destruction de la fenêtre d'une manière ou d'une autre.

Nous affichons la fenêtre avec un appel à `ewl_widget_show()`. Si `ewl_widget_show()` n'est pas appelée, rien ne s'affichera à l'écran. Tous les "widgets" sont masqués jusqu'a ce qu'ils soient explicitement affichés. L'opposition à cette fonction est `ewl_widget_hide()` qui cachera un widget à l'écran.

Exemple 10.4. The main container

```
/* 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);
```

Nous pourrions ranger tous nos "widgets" dans la fenêtre principale elle même, mais cela causerais des problèmes plus tard si nous voudrions changer quelque chose facilement, alors nous créérons une boite à l'intérieur de la fenêtre principale pour stocker tous nos "widgets".

Cela est fait par la création d'une boite verticale avec `ewl_vbox_new()`. La boite est ensuite prise et ajoutée à la liste des enfants de la fenêtre avec `ewl_container_child_append()`. Après l'avoir attachée à la fenêtre nous configurons la police de remplissage pour remplir la largeur et la hauteur avec `ewl_object_fill_policy_set()`, et affichons le widget avec `ewl_widget_show()`.

L'ordre avec lequel vous mettez vos widget dans les containeurs affectera la facon dont l'application sera affichée. Le premier "widget" ajouté sera le premier affiché. Puisque nous avons spécifié la création d'une boite verticale, nous commencerons par ajouter nos "widgets" du haut vers le bas de notre affichage.

Exemple 10.5. Créér la barre 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);
```

Le premier widget que nous placons est la barre de menu. Nous allons placer les éléments de la barre de menu après d'autres "widgets" mais nous devons placer la barre elle même en premier.

Les appels sont les même que ceux que vous avez vu plus tôt, nous nous déclarons à notre parent, configurons notre police de remplissage, affichons le widget. Celui qui n'a pas été vu plus haut est `ewl_object_alignment_set()`, cela configurera la facon dont le "widget" est aligné dans son containeur. Dans le cas présent nous utilisons EWL_FLAG_ALIGN_LEFT, mais aurions pu utiliser une autre des valeurs possibles :

- EWL_FLAG_ALIGN_CENTER
- EWL_FLAG_ALIGN_LEFT
- EWL_FLAG_ALIGN_RIGHT
- EWL_FLAG_ALIGN_TOP
- EWL_FLAG_ALIGN_BOTTOM

Le menu va dont s'aligner avec le coté gauche de la boite principale.

Nous spécifions ensuite l'espacement entre les éléments dans la boite de menu. Cela nous donnera un peu plus d'espacement entre nos éléments de menu et est réalisé par la fonction `ewl_box_spacing_set()`. Après avoir changé l'espacement, nous changeons les marges autour de la boite avec un appel à `ewl_object_padding_set()`, cela augmentera l'espace disponible autour de l'objet.

Exemple 10.6. Création de scrollpane

```
/* 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);
```

La scrollpane va être le parent de notre objet texte. Le scrollpane nous fournit les barres de défillement et les défillement lui même.

Le scrollpane est créé avec un appel à `ewl_scrollpane_new()`, et nous continuons en attachant le scrollpane à la boite principale, et configurons sa police de remplissage.

L'appel à ewl_scrollpane_[hv]scrollbar_flag_set() indique à Ewl comment les barres de défillement doivent se comporter. Les valeurs possibles sont :

- EWL_SCROLLBAR_FLAG_NONE
- EWL_SCROLLBAR_FLAG_AUTO_VISIBLE
- EWL_SCROLLBAR_FLAG_ALWAYS_HIDDEN

Uns fois les barres de défillements configurés nous demandons à Ewl d'afficher le "widget".

Exemple 10.7. Création de la zone de texte

```
/* 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);
```

La zone de texte sera responsable de contenir le texte que nous afficherons dans notre visionneur. Le "widget" est créé avec un simple appel à `ewl_text_new()`. Cela créera la zone de texte, mais avec un contenu vide. Comme pour la barre de menu, nosu augmentons les marges autours de la zone de texte pour disposer d'un peu plus d'espace entre la bordure de notre zone de texte et les autres éléments.

Exemple 10.8. Ajouter des éléments au 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);
}
```

Une fois que la zone de texte est créée nous pouvons continuer avec la créations de menus. J'ai fait cela dans un seul bloc pour limiter le nombre de déclaration au début de la fonction, cela n'est pas une nécessité.

Le menu est créé avec un appel à `ewl_imenu_new()`. Cette fonction prend deux paramètres, le premier est l'image à afficher avec ce menu, dans notre cas NULL, signifie que nous ne souhaitons pas inclure d'image. Le second paramètre est le nom de notre élément tel qu'il appraîtra dans la barre de menu.

Une fois le menu créé nous pouvons continuer à ajouter des entrées au menu avec `ewl_menu_item_new()`. Cette fonction demande deux paramètre, l'icon à afficher derrière cette entrée de menu, et le nom tel qu'il apparaîtra dans le menu.

Comme les éléments sont ajoutés au menu, nous réalisons un appel à `ewl_callback_append()` pour attacher l'appel EWL_CALLBACK_SELECT. La fonctin donnée sera exécutée lorsqu'un utilisateur cliquera sur une entrée du menu. Dans le cas de "open" nous avons passé le text_area à open pour nous permettre de modifier facilement son contenu.

D'autres menu peuvent être affichés en utilisant la même méthode, mais pour cette application, seul menu est nécessaire.

Exemple 10.9. Attacher les callbacks

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

Une fois que tout est configuré dans la fenêtre principale, nous attachons les callbacks que nous souhaitons recevoir. Dans notre cas nous nous attachons au callback EWL_CALLBACK_KEY_UP. Nous n'avons pas besoin de faire quoi que ce soit pour avoir le support de la roulette de la souris dans le scrollpane comme cela est configuré dans le scrollpane lui même.

Exemple 10.10. Callback destroy

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

Une fois la fenêtre rincipale fermée nous destruisons le widget qu'est la fenêtre principale par un appel à `ewl_widget_destroy()`. Une fois que la fenêtre est détruite, nous indiquons à Ewl qui nous souhaitons quitter en appelant `ewl_main_quit()`. Cela provoquera l'arrêt de la boucle principale Ewl et l'appel précédant à `ewl_main()` sera retourné.

Exemple 10.11. Callback d'ouverture de fichier du menu

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

    /* create the file dialog window */
    fd_win = ewl_window_new();
    ewl_window_title_set(EWL_WINDOW(fd_win), PROG " -- file dialog");
    ewl_window_name_set(EWL_WINDOW(fd_win), PROG " -- file dialog");
    ewl_window_class_set(EWL_WINDOW(fd_win), PROG " -- file dialog");
    ewl_object_size_request(EWL_OBJECT(fd_win), 500, 400);
    ewl_object_fill_policy_set(EWL_OBJECT(fd_win),
                EWL_FLAG_FILL_FILL | EWL_FLAG_FILL_SHRINK);
    ewl_callback_append(fd_win, EWL_CALLBACK_DELETE_WINDOW,
                                destroy_filedialog_cb, NULL);
    ewl_widget_show(fd_win);

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

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

    /* add a home button */
    home = ewl_button_new("Home");
    ewl_callback_append(home, EWL_CALLBACK_CLICKED, home_cb, fd);
```

```
    ewl_object_fill_policy_set(EWL_OBJECT(home), EWL_FLAG_FILL_HFILL);
    ewl_container_child_append(EWL_CONTAINER(fd), home);
    ewl_widget_show(home);

    ewl_widget_show(fd);
}
```

Si un utilisateur click sur l'entré "open" du menu fihcier, la fonction `file_menu_open_cb()` sera exécutée. Lorsque cela arrive, nous devons créer un dialogue de fichier pour que l'utilisateur selectionne le fichier à visionner.

Avec la même méthode que celle de la fenêtre principale nous créeons une fenêtre pour contenir le dialogue de fichier et configurons son titre, son nom et sa classe. Nous configurons aussi sa taille par défaut, sa police de remplissage et attachons un callback à la destruction de la fenêtre. Nous ajoutons ensuite une simple boite à la fenêtre qui contiendra le dialogue de fichier.

Une fois que la fenêtre est configuré, nous faisons l'appel Once the window is setup, we make the call to create the file dialog. This is done with a call to `ewl_filedialog_new()`, specifying the type of file dialog we wish to create. In this case we want a dialog to allow us to open a file, so we specify EWL_FILEDIALOG_TYPE_OPEN. We could have specified EWL_FILEDIALOG_TYPE_SAVE if we wished to use the dialog to save a file instead of open.

Nous procédons ensuite à la création de quelques boutons additionnels permettants à l'utilisateur de naviguer dans son dossier utilisateur à l'aide d'un simple click. Cela est fait par l'appel à `ewl_button_new()` et ajoutons ce bouton dans le dialog de sélection de fichier lui même.

Exemple 10.12. Callback de destruction de dialogue de fichier.

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

Lorsque nous n'avons plus besoin du dialogue de fichier nous retirons le "widget" de l'écran avec un appel à `ewl_widget_hide()`, puis une fois qu'il n'est plus sur l'écran nous le détruisons à l'aide de `ewl_widget_destroy()`.

Exemple 10.13. Callback du bouton ouvrir du dialogue de fichier

```
/* 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);
}
```

Ce callback sera exécuté lorsque l'utilisateur clickera sur le bouton ouvrir dans le dialogue de sélection de fichier, ou si celui-ci double-click sur un fichier dans le dossier. L'évènement passé (le paramètre ev) sera le chemin complet du fichier que l'utilisateur à sélectionné.

Dans notre cas, nous prenons ce fichier et le passons à la fonction pour lire le fichier et retourner son texte. Ensuite utilisant ce texte, nous appelons `ewl_text_text_set()` qui configurera le texte dans l'objet donné.

Une fois que l'utilisateur à fini sa sélection de fichier, le dialogue est masqué de l'écran.

Exemple 10.14. Callback du bouton "home" du dialogue de fichier

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

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

Si l'utilisateur clique sur le bouton "Home" dans le dialogue de fichier nous désirons montrer les fichiers de son dossier home. Nous configurons le dialogue de fichier comme données utilisateur au callback, puis nous réalisons un "cast" au Ewl_Filedialog et récupérons son dossier utilisateur depuis l'environnement. L'appel à `ewl_filedialog_set_directory()` change le dossier courant affiché à celui du dossier utilisateur.

Exemple 10.15. Lire le fichier texte

```
/* 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;
}
```

Il s'agit juste d'une routine simple pour prendre le fichier donné, le lire et stocker son contenu en mémoire. Probablement pas la meilleure méthode pour une application réelle, mais suffisant pour ce programme d'exemple.

Exemple 10.16. Callback d'appuis sur une touche

```
/* 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);
    }
}
```

La fonction `key_up_cb()` sera appelée dès qu'un utilisateur appuiera sur une touche sur le clavier. Le callback recevra une structure Ewl_Event_Key_Down contenant l'information sur la touche elle même. Dans notre cas nous n'avons besoin que de lu champ "keyname" qui est le nom de touche pressée.

Si l'utilisateur tappe "q" nous appelons simplement le callback destroy.

Les touches "Left", "Right", "Up" et "Down" représentent les touches de direction du clavier de l'utilisateur. Si l'une de ces touches est pressée nous forcons le scrollpane à défiller dans un certain sens.

Dans le but de manipuler le scrollpane, nous devons savoir où il se trouve dans le ficheir et la distance que chaque montée/descente doit parcourrir. Heuresement Ewl rend cela très simple. L'appel à ewl_scrollpane_[hv]scrollbar_value_get() retournera la valeur courante de la barre de défillement. Cela est une valeur double comprise dans la plage [0, 1]. Une valeur de 0 signifie que la barre de défillement est en haut et une valeur de 1 qu'elle est en bas. La droite et la gauche fonctionnent de la même facon sauf que 0 est la gauche et 1 la droite.

La seconde partie de l'information est obtenue au travers de l'appel à ewl_scrollpane_[hv]scrollbar_step_get(). Le "step" est la distance que le scrollpane parcourera en un seul déplacement. Donc en utilisant ces deux valeurs nous

pouvons déplacer la barre de défillement dans la direction voulue avec un appel à ewl_scrollpane_[hv]scrollbar_value_set().

Exemple 10.17. Compilation

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

Compiler une application ewl est aussi simple qu'appeler ewl-config et obtenir les --cflags et --libs.

Voilà. Avec cela vous devriez obtenir une application Ewl complète et fonctionnelle incluant des menus, un dialogue de sélection de fichier et une zone de texte avec des ascenseurs verticaux et horizontaux. Cet exemple éfleure simplement la puissance contenue dans la boite à outils Ewl qui contient bien d'autres "widgets" prêts à l'emploi.

Chapitre 11. Evoak

Evoak est un serveur de canvas. Il est similaire au serveur X qui réalise les opérations graphiques et d'affichage. Evoak sert un canvas unique à plusieurs applications (clients) autorisant chaque client à manipuler ses objets sur le canvas.

Recette : Client hello Evoak

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

Cette recette est une introduction très simple au monde de la programmation Evoak. Perpétuant les grandes traditions anciennes, elle montre la version Candienne du 'Hello World' sur un canvas Evoak.

Exemple 11.1. Inclusions et Pré-déclarations

```
#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 *);
```

Nous devons évidemment inclure le fichier en-tête Evoak, et celui de Ecore qui est nécessaire pour avoir accès aux fonctions de callbacks.

Exemple 11.2. 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 a besoin d'une configuration initiale à l'aide d'un appel à `evoak_init`. Cela configurera les libraires internes et le nécessaire pour Evoak.

Si Evoak se charge correctement, nous prennons en charge deux callbacks, le premier est pour les informations sur le canvas et le second si nous sommes déconnectés du serveur Evoak. Cela sera expliqué plus tard lorsque les callbacks actuels seront affichés.

Une fois les callbacks en place nous devons nous connecter au canvas du serveur Evoak. Cela est fait au travers d'un appel à `evoak_connect`. Les paramètres passés à `evoak_connect` sont: le serveur auquel se connecter, le nom du client et la classe du client. Si le premier argument est NULL, comme c'est le cas dans l'exemple, le serveur Evoak par défaut sera aussi connecté. Le second argument passé à `ecore_connect` est le nom du client, cette valeur doit être unique car elle est utilisée pour distinguer un client des autres. Le dernier argument, la classe, est le type de client, quelques valeurs possibles sont : "background", "panel", "application" ou "custom".

Si l'appel à `evoak_connect` échoue, la valeur NULL est renvoyée. Donc une fois que nous recevons un objet Evoak, nous entamons la boucle principale. Une fois qu'ecore à terminé nous appelons `evoak_disconnect` pour nous déconnecter du serveur Evoak.

Nous terminons par l'appel `evoak_shutdown` pour nettoyer ce que nous avons créé.

Exemple 11.3. Callback d'informations sur le 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;
}
```

Un callback d'informations sur le canvas sera fait lorsque notre client recevra des informations concernant le canvas serveur Evoak. Avec cet information sur le canvas, nous pouvons procéder au paramétrage du contenu de nos clients. Cela est contenu à l'interrieur d'un drapeau setup_called car nous ne souhaitons l'initialiser qu'une seule fois.

Exemple 11.4. Callback disconnect

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

Le callback disconnect sera appelé lorsque le client sera déconnecté du serveur Evoak. Dans ce cas la solution simple de la fermeture est utilisée.

Exemple 11.5. Routine de configuration

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

    evoak_freeze(ev);
```

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

    evoak_thaw(ev);
}
```

La routine de configuration sera appelée une seule fois pour configurer l'affichage de notre client. Pour cet exemple, le client ne dessinne qu'un texte 'Hello Evoak, eh'.

La première chose que nous appelons est `evoak_freeze`, cela devrait nous mettre à l'abris de callbacks non souhaités pendant que nous configurons notre interface. A la fin de la fonction nous appelons la réciproque `evoak_thaw` pour désactiver le freeze précédent.

Nous commencons ensuite la création d'un objet avec `evoak_object_text_add` et prenons cet objet, et configurons la police, la couleur et le contenu du texte avec les appels à `evoak_object_text_font_set`, `evoak_object_color_set`, et `evoak_object_text_text_set` respectivement.

Exemple 11.6. Compilation

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

Comment pour beaucoups autres librairies basées sur EFL, la compilation d'un application Evoak est simplifiée par l'appel au programme evoak-config en y ajoutant les options --cflags et --libs.

Voila, ce fut une introduction très simple à Evoak et la surface reste inexplorée vis a vis du potentiel disponnible pour les applications clientes.

Chapitre 12. Emotion

Emotion est une librairie d'objets vidéo & et média développée dans le but d'être interfacée avec Evas et Ecore pour fournir des objets "vidéo" et "son" pouvant être déplacés, redimensionnés et positionnés comme n'importe quel autre objet, mais en plus ils peuvent jouer des vidéos et du son et peuvent être contrôlés par une API de contrôle de haut-niveau autorisant un développeur à créér avec lui un système multi-média avec peu d'efforts. Emotion fourni un système de couche de décodage où un module de décodage peut être chargé séparemment pour fournir de multiples ressources de décodage à Emotion. Emotion dispose actuellement d'un module utilise XINE comme décodeur, autorisant la lecture des DVDs, MPEGs, AVIs, MOVs, WMVs et bien plus. Son programme de test est déja un lecteur DVD très pratique (avec beaucoups des contrôles de l'interface) et peut jouer des vidéos avec de la semi-transparence et bien plus.

Recette : Un lecteur de DVD rapide avec Emotion

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

Pour montrer à quel point il est facile de mettre un objet vidéo DVD, VCD ou autre dans un "canvas", regardez le programme suivant. Ceci est un lecteur de DVD complet, mais très simpl. Il dispose de contrôles souris limités, aucune prise en charge du changement de ratio, etc. Il fait en tout 55 lignes de code C.

Le code ci-dessous et la recette suivante peuvent être compilés en utilisant :

Exemple 12.1. Compilation

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

Exemple 12.2. Lecteur de DVD en 55 lignes de code

```
#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();
}
```

Maintenant nous avons une introduction très simple à Emotion. Cet extrait de code peut facilement être complété pour travailler avec n'importe quel format supporté par emotion, aussi bien que gérer les ratios, la navigation au clavier et bien plus.

Recette : Lecteur de vidéo étendu avec Emotion

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

Extension basée sur notre précédente recette, nous pouvons transférer la gestion du redimensionnement à emotion proprement (tiens compte de l'aspet ratio). Expanding on our previous recipie, we can make emotion handle being resized properly (which maintaining aspect ration),

Exemple 12.3. Lecteur de vidéo 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();
}
```