

Apprenez les bases du C en réalisant un économiseur d'écran avec Xscreensaver

Learn the basics of the C Programming Language, by creating a screen-saver for Xscreensaver

Translation in Progress

This article has for purpose to propose to people who are not totally familiarized with C, to exercise with Xscreensaver, while learning some basics of the Xlib library.

This article is there for people who are beginners with C, to train their skills while creating a screen-saver with Xscreensaver, and the Xlib library.

Xscreensaver is one of the most popular screen-saver distributions for Linux. These screen-savers can be in 2D, using the Xlib library, or in 3D using OpenGL. (Only 2D will be seen in this article.)

Xscreensaver is probably one of the most popular screen-saver distribution for the Linux, and OSX systems. These screen-savers can be written in pure 2D, by using the Xlib library, or with OpenGL to make 3D savers with GLX. (We will see only the 2D in this article.)

If you want to create a screen-saver for Xscreensaver with OpenGL, I would recommend to use OpenGL ES so that it can be compatible with the RaspberryPi.

In case you would like to create a screen-saver for Xscreensaver with OpenGL, we would recommend to use OpenGL ES, in order to keep your compatibility with the RaspberryPi's.

At the date of writing this article, the last version of Xscreensaver is version 6.09, and the screen-savers are gathered in the `hacks/` directory (and those in 3D are in `hacks/glx/`).

At the date this article was written, the last version of Xscreensaver is the version 6.09, and the screen-savers are located inside the `hacks/` directory (and those for 3D are located inside `hacks/glx/`.)

To create your own screen-saver (also called demo), you can just make the `configure` step at the root of the sources, and then `make` in the `utils/` directory. Then the demos of the `hacks/` directory can be compiled individually, like for example `munch.c`, with `make munch`, then this demo can be visualised locally inside the sources with `./munch`. This demo will then start in its own window. You can make the same with your own demo, in order to test the results. To compile your demo, you can either edit the `Makefile`, to add your own demo next to the others, or you can copy the command that you saw to compile another already existing screen-saver.

To create your own screen-saver (also called demo), you can just make the `configure` step at the root of the sources, and then `make` in the directory `utils/`. Then the demos of the `hacks/` directory can be then compiled individually, like for example `munch.c`, with `make munch`, then this demo can be visualised locally inside the sources with `./munch`. This demo will then start in its own window. You can make the same with your own demo, in order to test the results. To compile your demo, you can either edit the `Makefile`, to add your own demo next to the others, or you can copy the command that you saw to compile another already existing screen-saver.

Le fichier d'entête à inclure au début de votre économiseur d'écran est :

The .c header that you have to include at the beginning of the source of you screen-saver is:

```
#include "screenhack.h"
```

L'inclusion d'un fichier d'entête est la méthode standard en C pour inclure des prototypes de fonction à utiliser, ainsi que les types de données associées.

The include of a .h header in .c is the standard method to include the prototypes of the functions that you want to use, and also the data-types of the associated data.

Pour votre économiseur, pas besoin d'inclure une fonction `main()`, qui est généralement le point de départ dans les programmes C, mais ici vous devrez inclure cinq fonctions de rappel (callback) : "init", "draw", "reshape", "event" et "free". Chacune de ces fonctions devra être préfixée avec le nom de votre démo, et ce préfix devra être déclaré avec la macro `XSCREENSAVER_MODULE()`.

In your saver, no need to include a main() function, which is usually the entry point to start the C program, but here you will have to write five callback- functions: "init", "draw", "reshape", "event" and "free". Each of these functions will have to be prefixed with the name of your demo, and this prefix will have to be registered with the XSCREENSAVER_MODULE() macro.

```
/* Entry Points */

static void *
my_xss_init(Display *display, Window window) { ... }

static unsigned long
my_xss_draw(Display *display, Window window, void *closure) { ... }

static void
my_xss_reshape(Display *display, Window window, void *closure,
               unsigned int w, unsigned int h) { ... }

static Bool
my_xss_event(Display *display, Window window, void *closure,
             XEvent *event) { ... }

static void
my_xss_free(Display *display, Window window, void *closure) { ... }

XSCREENSAVER_MODULE("my_xss", my_xss)
```

Les Fonctions en C

Les fonctions en C, comme dans les autres langages de programmation, regroupent des séries d'instructions dans des blocs de code. Les blocs de code en C sont regroupés à l'intérieur d'accolades `{ ... }`.

The C functions, like in other programming languages, are grouping together series of instruction with blocks of source code. These blocks of code, with C are grouped inside curly brackets { ... }.

Voici une fonction simple qui réalise une addition entre deux nombres entiers passés en paramètre, et qui retourne le résultat :

Here is an example of a simple function, which creates an addition between two integer numbers, which are provided as parameters to the function between the round-brackets:

```
int add(int a, int b) {
    int c;
    c = a + b;
    return c;
}
```

La première ligne correspond au prototype de la fonction, qui inclut le type de retour, le nom de la fonction, suivit des paramètres de la fonction, eux aussi précédés par leur type. Le corps de la fonction placé entre des accolades contient toutes les instructions qui seront exécutées quand la fonction sera appelée.

The first line is equivalent than the function prototype, which also includes the returned type, at the beginning, followed by the name of the function, and with the function parameters, between the round-brackets, with the type of the parameters before their names.

Les Variables en C

C Variables

En C, les variables doivent d'abord être déclarées avec leur type avant d'être utilisées. C'est ce qu'on observe dans la fonction précédente avec la ligne de code `int c;`.

Variables in C should be declared first, with their type before their name, before they can be used. This is what we can see in the previous function, at the first line, with `int c;`.

Les Structures en C

C Structures

Les Structures en C regroupent plusieurs éléments similaires à des variables. Chacuns de ces éléments sont appelés des champs, et dans la déclaration du type de la structure chaque nom de champ doit être précédé par son type.

C structures group together several elements similar than variables. Each of these elements are called fields, and in the declaration of the type of the structure, each field name have to be preceded by its type.

Pour réaliser un économiseur d'écran avec Xscreensaver vous devez regrouper toutes les variables d'état dans une structure principale. Ces structures sont nommée `state` par convention dans les économiseurs d'Xscreensaver.

In order to create a screen-saver for Xscreensaver, you should first group together of the state-variables, in a main structure. There structures are called `state`, in the convention of the Xscreensaver's savers.

```
/* State struct */

struct state {
    Display *display;
    Window window;
    GC gc;
    int width;
    int height;
    // ajoutez ici vos propres éléments
    // add your own elements here
};
```

Ici, Les trois type `Display`, `window` et `GC` correspondent à des types de la bibliothèque Xlib. Cette bibliothèque permet de communiquer avec le serveur X.org, qui est responsable de l'affichage graphique sous Linux.

La variable `display` est la variable principale qui permet la communication entre l'application cliente (le logiciel avec une fenêtre) et le serveur X (qui réalise l'affichage, juste au-dessus de l'OS). La variable `window` contient l'identifiant de la fenêtre où dessiner les éléments. La variable `gc` contient un contexte graphique (graphics context), qui est nécessaire pour réaliser des opérations de dessin.

Here the three types `Display`, `Window` and `GC` are the main types of the Xlib library. This library makes you possible to communicate with the X.org server, which makes it possible to draw on screen under the Linux System.

The `display` variable is the main variable to establish a communicate between a client application, and the graphics-server. The client application is the application which will draw the interface of the program inside a window. And the X server creates the display, on top of the OS.

The `window` variable contains the ID of the window in which the graphics primitives will draw the elements the the window will display to the user.

The `gc` variable provides the elements that you can use as graphics context.

La déclaration du type de la structure `state` ne provoquera pas la réservation d'un espace en mémoire pour contenir cet élément. On va donc allouer cet espace en mémoire dans la fonction "init" à l'aide de la fonction C `malloc()`, comme ceci :

The declaration of the type of the structure `state` is not enough to create a space in memory to hold the contents of this variable. We then have to allocate a space in memory to contain this element. We then allocate this space in memory in the function "init", with the help of the `malloc()` C function, like this:

```
struct state *st = (struct state *) malloc(sizeof(struct state));
```

Après cette allocation, tous les champs pourront être initialisés.

Même si toutes les fonctions de rappel transmettent les variables `display` et `window`, les inclure dans la structure `state` peut être une bonne idée pour pouvoir les passer plus facilement à des sous fonctions éventuelles.

```
st->display = display;
st->>window = window;
```

La structure `state` ayant été ici allouée dynamiquement, on accède à ses champs avec une petite flèche `->`. Lorsqu'une structure ne provient pas d'une allocation dynamique (variable automatique), les champs sont accédés avec un point : `st.field`.

La première opération à inclure dans les autres fonctions de rappel, est la coercition (cast) du type de la structure `state`, car le type par défaut proposé par l'API d'Xscreensaver est un type générique. Ce type générique est ici un pointeur générique `void *closure` qui peut pointer vers l'espace mémoire de n'importe quelle structure définie par les contributeurs à Xscreensaver.

```
/* state cast */
struct state *st = (struct state *) closure;
```

La coercition est réalisée en indiquant le type cible entre parenthèses, devant la variable.

Voici ci-dessous un template d'économiseur qui n'affichera qu'une fenêtre vide, mais que vous pouvez utiliser comme point de départ, et pour tester que la compilation et l'exécution fonctionnent correctement (compilez ce code comme expliqué au début de l'article).

```
#include "screenhack.h"
```

```

/* State struct */

struct state {
    Display *display;
    Window window;
    GC gc;
};

/* Entry Points */

static void *
my_xss_init(Display *display, Window window) {
    struct state *st = (struct state *) malloc(sizeof(struct state));
    return st;
}

static unsigned long
my_xss_draw(Display *display, Window window, void *closure) {
    return 100000;
}

static void
my_xss_reshape(Display *display, Window window, void *closure,
               unsigned int w, unsigned int h) { }

static Bool
my_xss_event(Display *display, Window window, void *closure,
             XEvent *event) { return False; }

static void
my_xss_free(Display *display, Window window, void *closure) {
}

/* Defaults and Options */
static const char *my_xss_defaults[] = {
    0
};

static XrmOptionDescRec my_xss_options[] = {
    { 0, 0, 0, 0 }
};

XSCREENSAVER_MODULE("my_xss", my_xss)

```

Les tableaux pour les paramètres par défaut et les options peuvent être laissés vides dans un premier temps.

Si la fenêtre est trop grande sur l'écran d'un ordinateur portable, vous pouvez spécifier la taille de la fenêtre comme ceci :

```
./myxss -geometry 400x300
```

Le callback "reshape" est appelé lorsque la fenêtre de la démo est redimensionnée. Il est également appelé une première fois au début de l'exécution du programme, juste *après* le callback d'initialisation "init". Aussi si vous avez besoin de ces dimensions pour initialiser votre démo dans le callback d'initialisation, la largeur et la hauteur de la fenêtre peuvent être récupérées avec la fonction `XGetWindowAttributes()`.

The "reshape" callback is called when the window of the demo is reshaped. This callback is also called once at the beginning, when the demo is started, so you can use it to initialise the size of the window in which you can draw your saver.

```

{
    XWindowAttributes xgwa;
    XGetWindowAttributes(display, window, &xgwa);
}

```

```

    st->w = xgwa.width;
    st->h = xgwa.height;
}

```

Le caractère '&' permet de transmettre l'adresse en mémoire de la structure, lorsque cette structure n'a pas été allouée dynamiquement.

Vous pourriez utiliser votre première primitive de dessin `XFillRectangle()` comme ceci, à partir du template vide précédent :

You could use your first drawing primitive `XFillRectangle()` like this, from the previous template:

```

static void *
my_xss_init(Display *display, Window window) {
    struct state *st = malloc(sizeof(*st));
    unsigned long fg;
    st->display = display;
    st->window = window;
    st->gc = XDefaultGC(display, 0);
    fg = XWhitePixel(display, 0);
    XSetForeground(display, st->gc, fg);
    return st;
}

static unsigned long
my_xss_draw(Display *display, Window window, void *closure) {
    struct state *st = (struct state *) closure;
    XFillRectangle(st->display, st->window, st->gc,
        20, 20, 100, 60);
    return 200000;
}

```

Le fond étant noir par défaut, ici nous sélectionnons le blanc comme couleur de dessin.

Le nombre retourné par la fonction "draw" correspond au nombre de microsecondes à attendre avant le prochain appel à la fonction draw. Ainsi avec 200000, nous aurons 5 frames par seconde.

Pour dessiner avec une autre couleur, il faut allouer une couleur dans le serveur X avec la fonction `XAllocColor()`, puis sélectionner cette couleur avec `XSetForeground()`.

```

int screen_number = 0;
Colormap colormap = XDefaultColormap(display, screen_number);
XColor color;
color.red = 65535;
color.green = 0;
color.blue = 0;
XAllocColor(display, colormap, &color);
XSetForeground(display, st->gc, color.pixel);

```

65535 est la valeur maximale que l'on peut obtenir avec une variable représentant un entier positif sur 16 bits.

Primitives de dessin

```

XDrawPoint(st->display, st->window, st->gc, x, y);
XDrawLine(st->display, st->window, st->gc, x1, y1, x2, y2);
XDrawRectangle(st->display, st->window, st->gc, x, y, width, height);
XDrawArc(st->display, st->window, st->gc, x, y, width, height, angle1, angle2);

```

Pour dessiner un cercle, vous pouvez spécifier 0 et 360*64 pour angle1 et angle2.

```

XFillArc(st->display, st->window, st->gc, x, y, width, height, 0, 360*64);

```

Ces primitives de dessin sont basiques, cependant plein de choses très intéressantes peuvent être réalisées avec celles-ci. Cette liste est non-exhaustive, vous en trouverez bien plus au chapitre 8 de cette documentation sur la Xlib :

- <https://tronche.com/gui/x/xlib/>

Les Options

Pour un exemple d'utilisation des options que l'on peut passer en ligne de commande lors du lancement d'un économiseur, et de leurs valeurs par défaut, je vous recommanderais de regarder le code source de la démo `popsquares.c`.

Les événements

Pour un exemple de récupération des touches du clavier dans le callback "event", vous pouvez jeter un oeil à la démo `xmatrix.c`. Et vous trouverez dans la démo `pong.c`, qui est une démo jouable, comment utiliser l'évènement `MotionNotify` pour récupérer les mouvements de la souris. ("julia" et "attraction" sont également des démos interactives, mais avec un clic de la souris.)

Les évènements de type `XEvent` peuvent être accédés avec `event->xkey` ou `event->xbutton` ou autre, en fonction du type de l'évènement, car le type `XEvent` est un type union en C (défini dans `/usr/include/X11/Xlib.h`). Un type union rassemble plusieurs types en un seul, et qui peuvent être utilisés de manière alternative. Le premier champ de tous les types d'évènements est toujours le champ `int type`, qui permet de savoir lequel des types de l'union utiliser.

Comme vous pouvez le voir dans les démos d'`Xscreensaver`, on sélectionne et on utilise `event->xkey` après avoir vérifié que `event->type` a pour valeur `KeyPress`, et `event->xbutton` lorsque `event->type` a pour valeur `ButtonPress`.

Les tableaux

Si vous souhaitez utiliser des tableaux C dans votre démo, vous pouvez réserver initialement un espace mémoire de taille fixe comme ceci :

```
// tableau de 10 entiers
{
    int elems[10];
}
```

Ou bien allouer dynamiquement de la mémoire à l'exécution avec `malloc()` ou `calloc()` :

```
{
    int *elems;
    elems = calloc(10, sizeof(int));
    elems[0] = 5;
}
```

(Tapez `man malloc` pour plus de détails.)

Les pointeurs

Ce que retournent les fonctions de type `malloc()` est une adresse mémoire. Ces adresses sont stockées dans des variables de type pointeur, que l'on reconnaît au caractère étoile (astérisque), dans le type de la déclaration de l'expression `int *elems;`.

Idées en vrac

Voici quelques idées d'économiseurs simples à réaliser si vous souhaitez vous exercer avec le C, et si vous manquez d'inspiration.

- Des éléments ressemblant aux avatars de github, constitués de carrés sur une grille symétrique de 5 par 5, et qui évoluent dans la fenêtre.
- Des cercles représentant des planètes avec des satellites qui gravitent autour.
- Des cercles qui suivent des chemins constitués de courbes de bezier.
- Utiliser le callback "event" pour ajouter des tours qui tirent sur les cercles de l'idée précédente, ce qui ferait un mini-jeu de type "tower-defense".
- Une démo avec des éléments géométriques simples qui s'inspire du mouvement pop-art, avec les peintres tels que Victor Vasarely et Bridget Riley (et notamment leurs œuvres Nataraja et Orion, qui sont simples et intéressantes visuellement à reproduire).
- Des petites entités représentées par de petits carrés qui évoluent toutes seules dans la fenêtre, et collectent éventuellement des éléments qui apparaissent aléatoirement.

Pour cette dernière idée, une simple structure avec des coordonnées x et y, et avec une direction peut suffire.

L'étape de la conception

Lors de l'étape de conception, je vous recommanderais d'utiliser une feuille et un crayon. De faire des croquis pour rechercher des idées, puis définir plus précisément ce que vous voudriez obtenir.

Vous pourriez également réaliser un schéma sur la manière dont vous pensez organiser et structurer votre code.

Ensuite pour passer à la réalisation, je vous recommanderais de procéder de manière itérative, en commençant par une version extrêmement simple, à laquelle vous ajouterez les détails successivement, par étapes.