Mode d'emploi d'easyX11 (pour les programmeurs)

Résumé :
easyX11 peut être vu comme un mini *window manager* fonctionnant à l'intérieur d'une seule fenêtre X-windows avec deux types de sous-fenêtres et six applications pré-programmées (cadrans, ascenseurs, menus *popup*, sélecteur de fichier, visualisateur de texte et d'images). Il délivre le programmeur C de la plupart des problèmes de gestion graphiques proches du matériel.

Contenu de ce document :

Ce manuel est divisé en trois parties : Les fonctions de base sont celles strictement nécessaires pour réaliser une application graphique interactive, les fonctions avancées permettent de jouer avec les fontes, les curseurs et le serveur X-windows, enfin la boîte à outil contient six applications (ascenseurs, cadrans/boutons, menus, visu de texte, visu d'image et sélecteur de fichier). Ces applications sont, en fait, écrites avec les fonctions de base de easyX11.

utilisation :

Un source d'application easyX11 doit contenir les déclarations et prototypes de easyX11, il faut donc avoir :

```
#include "easyX11.h"
```

en début de programme. Cela suppose qu'une copie (ou mieux un lien symbolique vers) du fichier `easyX11.h` soit dans votre *directory*. Vous pouvez aussi bien mettre le chemin complet. Le fichier est disponible sur `/sabrina/local/bin`, `/rosanna/local/bin`, `/wanda1/local/bin` et `/corona2/bin`.

A la compilation, une application easyX11 doit inclure les librairies easyX11, X11 et m (math). L'ordre de compilation type est :

```
cc toto.c easyX11.o -leasyX11 -lX11 -lm -o toto
```

Ceci suppose toujours que `libeasyX11.a` est dans le *directory* `/lib`. C'est le cas sur les machines en système V (Rosanna, Wanda et Adele). Sinon mettre le chemin par l'option `-L/usr/X11R5/lib` avant l'option `-l`. Sur la famille Sabrina et sur la famille Corona, la librairie est dans `/usr/X11R5/lib`. (et aussi `/usr/X11R5/staticlib` sur Sabrina).

A l'exécution, un programme easyX11 s'attend à trouver quatre variables d'environnement : **DISPLAY** contenant le nom de votre terminal et le numéro d'écran (p.ex. `harpo:0` sur mon terminal). Sur l'écran d'une station `:0` est plus rapide que <nom>:0.

EASYX11FILE (facultatif) contenant le nom d'un fichier de fonte easyX11, ce fichier doit vous appartenir (pour pouvoir changer de mode graphique par la commande `mode` et de fonte par la commande `fonte`). Je vous suggère de recopier la fonte `latin9x15` par défaut dans un fichier `.easyX11rc` de votre *home directory*. Attention, la fonte utilisée ne doit pas dépasser 32x64 en dimensions ! Si la variable d'environnement n'est pas définie, le programme utilise une fonte interne par défaut (latin1 en 9x15) et un mode par défaut (bitmap, diffuse ou direct en fonction du matériel détecté).

EASYX11DIR (facultatif) contenant le chemin (terminé par / et de préférence absolu) d'un *directory* contenant les fontes easyX11. Elles sont en particulier dans les *directories* `/rosanna/local/bin/EASYX11.FONTES`, `/sabrina/local/bin/EASYX11.FONTES` et `/corona2/bin/EASYX11.FONTES`.

EASYX11GAIN (facultatif) contient les coefficients (séparés par des virgules) d'une correspondance polynomiale du second degré entre les luminances "logicielles" de 0 à 255 et les luminances "matérielles" du seuil à la saturation des phosphores de l'écran. Ces coefficients peuvent être déterminés grâce à l'utilitaire `easyX11gain`.

EASYX11LOG (facultatif) donne un nom de fichier où seront enregistrées toutes les manipulations de l'utilisateur comme les déplacements de souris, entrées au clavier...

EASYSX11REPLAY (facultatif et exclusive de la précédente) donne un nom de fichier enregistré par l'utilisation d'EASYX11LOG pour rejouer les manipulations de l'opérateur en mode automatique. Pendant que les opérations se rejouent, on peut faire une pause en enfonçant la touche espace, on peut alors soit reprendre en enfonçant la touche [retour] soit interrompre et reprendre la main en enfonçant la touche [escape]. Pour des raisons techniques complexes, il n'est pas possible d'interrompre pendant une lecture par readbox(), ni de rejouer des interruptions par eventX11() ou startX11(). Enfin il est possible d'éditer le fichier d'EASYX11LOG voire d'insérer des pauses (caractère P suivi d'un commentaire affiché à la console jusqu'à la fin de la ligne).

Enfin comme certaines implémentations de X-windows ont des erreurs ou de différentes interprétations de la norme X11, il peut être nécessaire de positionner une ou plusieurs des variables suivantes (leur valeur n'a pas d'importance) :

EASYX11SUNBUG S'il n'est pas possible d'utiliser le mode "*focus on root*" avec votre window manager, et qu'il est donc impossible de rentrer des informations au clavier (essayer par exemple de modifier la **fonction de filtrage** dans le menu **données**).

EASYX11STARDENTBUG Si les rectangles autour des boutons en haut de la fenêtre easyX11 sont incomplètement tracés.

EASYX11IBMBUG Si au contraire des triangles noirs ou blancs apparaissent dans les boutons de haut de fenêtre et gênent la lecture des intitulés de boutons.

EASYX11NCDBUG Cette erreur est moins facile à tester, il faut recouvrir partiellement les quatre coins de la fenêtre easyX11 par quatre autres fenêtres de telle façon que les bords horizontaux des fenêtres soient exactement sur la même ligne. Si le serveur X "plante" quand on remonte la fenêtre easyX11, c'est qu'il faut positionner cette variable.
ATTENTION: ne positionnez pas toutes ces variables "au cas où", en particulier la dernière ralentit considérablement les opérations de rafraîchissement de la fenêtre.

couleurs :
Les sous-fenêtres des deux types utilisent des couleurs easyX11 pour les textes et les graphismes. Ces couleurs sont affichées telles quelles même si le mode graphique est MONO (niveau de gris) ou PSEUDO (fausses couleurs). Elles sont en nombre très réduit pour permettre l'utilisation du reste de la table des couleurs. En particulier, il ne sera donné suite à aucune demande de couleurs supplémentaires !
Le code des couleurs est le suivant :

constante définie dans easyX11.h	valeur	apparence à l'écran
TRANSPARENT	0	invisible sur graphique
BLEU	1	bleu plein saturé
VERT	2	vert plein saturé
ROUGE	3	rouge plein saturé
BLANC	4	blanc
NOIR	5	noir opaque
GRIS	6	gris à 50%
	7	blanc sur graphique, couleur de fond sur boîte

boutons de souris :
Les combinaisons boutons/touche de contrôle ont les effets suivants :

Bouton (plus touche éventuelle)	effet si enfoncé	effet si relâché
gauche	envoie le code 1 / début de copie(1)	envoie le code −1 / fin de copie(1)
gauche+Shift	envoie le code 2	envoie le code −2
gauche+Ctrl	envoie le code 3	envoie le code −3
gauche+Shift+Ctrl	remonte la sous-fenêtre(2)	(sans effet)
centre	envoie le code 2 / colle(3)	envoie le code −2
centre+Shift	envoie le code 4	envoie le code −4
centre+Ctrl	envoie le code 5	envoie le code −5
centre+Shift+Ctrl	commence à déplacer la sous-fenêtre(2)	finit de déplacer la sous-fenêtre
droite	envoie le code 3 / fin de copie(1)	envoie le code −3
droite+Shift	envoie le code 6	envoie le code −6
droite+Ctrl	envoie le code 7	envoie le code −7
droite+Shift+Ctrl	enfonce la sous-fenêtre(2)	(sans effet)

(1) sur les boîtes et pendant readbox et textselect.

(2) rafraîchit toute la fenêtre si en dehors de toute sous-fenêtre. Il est possible de copier coller avec des applications X-Windows, le mécanisme utilisé est celui de la sélection, et en secours, celui du *cut-buffer*.
(3) pendant readbox.

divers :
Les dimensions de l'écran, de la fenêtre et de la fonte sont dans des variables entières déclarées dans `easyX11.h` :
eX_LARG= largeur de l'écran en pixels, **eX_LONG=** hauteur de l'écran, **eX_larg=** largeur de la fenêtre easyX11, **eX_long=** hauteur de la fenêtre easyX11, **eX_largchar=** largeur max d'un caractère, **eX_longchar=** hauteur max d'un caractère, **eX_basechar=** position de la ligne de base des caractères (par rapport au coté haut).

FONCTIONS DE BASE:

Avant d'appeler toute fonction de easyX11, il faut appeler la fonction :

```
int openX11(int pos_horiz, int pos_vert, int largeur, int hauteur, int f_touche(), int f_resize(), int override);
```

Largeur et hauteur sont en pixels, Pos_horiz (resp. pos_vert) sont mesurées à partir du bord gauche (resp. haut) de l'écran s'ils sont >0, à partir du bord droite (resp. bas) s'ils sont <0 et la fenêtre est positionnée au milieu entre les deux bords s'ils sont =0.
f_touche est une fonction (optionnelle, on peut passer `NULL`) qui est appelée avec comme argument une chaîne de caractères correspondant à chaque touche enfoncée quand on enfonce une touche (en dehors de la fonction readbox, voir plus bas). La chaîne de caractère ascii comporte plusieurs caractères pour les touches spéciales, le premier caractère est un *escape* (27=033=0x1B) les codes sont ceux obtenus par scanf ou xterm A L'EXCEPTION de [Home] et [End] pour lesquels le code ambigu "\33[" a été remplacé par "\33[3~" et "\33[4~" resp. les codes non supportés sont formés de 4 caractères (les 32 bits du key_sym en low order first).

Touche	code
↑	\33[A
↓	\33[B
→	\33[C
←	\33[D
Ins	\33[2~
Home	\33[3~
End	\33[4~
PgUp	\33[5~
PgDn	\33[6~
F1	\33[11~
F2	\33[12~
F3	\33[13~
F4	\33[14~
F5	\33[15~
F6	\33[17~
F7	\33[18~
F8	\33[19~
F9	\33[20~
F10	\33[21~
F11	\33[23~
F12	\33[24~

f_resize est une fonction (optionnelle, on peut passer `NULL`) qui est appelée avec comme arguments la nouvelle largeur et la nouvelle hauteur à chaque fois que l'on change la taille de la fenêtre.
override est un entier qui, s'il est non nul, empêche le *window manager* de maîtriser la fenêtre (il n'y a alors plus de cadre, il est impossible de la bouger, de modifier sa taille ou son empilement...)
Il est possible d'appeler de nouveau après `terminateX11()` ou `closeX11()` avec éventuellement de nouvelles dimensions, mais il4faut alors rouvrir toutes les sous-fenêtres graphiques et toutes les boîtes.

Il est aussi possible de modifier avant(*) appel le nom de la fenêtre et de son icône, en modifiant les pointeurs eX_name et eX_icon_name.

Si l'on ne veut pas ouvrir de fenêtre (par exemple pour lancer easyX11 sur une fenêtre crée par un autre programme ou sur la *root window*), il faut mettre largeur ou hauteur à zéro. Attention cependant : cette manipulation n'est pas possible dans les modes graphiques xxx–LOCAL (il faudrait passer à easyX11 un colormap attribué à la fenêtre)
Un programme easyX11 fonctionne généralement de façon asynchrone, c'est-à-dire que ce sont les événements déclenchés par l'opérateur qui provoquent (avec la souris et le clavier) des appels de fonctions définies par le programme. Il est possible pour des applications très rudimentaires de fonctionner de manière traditionnelle, sans donner la main au serveur X-window, mais il n'y a pas alors de rafraîchissement de la fenêtre en cas de besoin, ni de possibilité de récupérer les actions de l'utilisateur sur le clavier ou la souris. Il est aussi possible d'interroger explicitement le serveur pour des applications de style "temps réel" mais ce genre de pratique est très coûteuse sauf pour des applications très calculatoires, et cette fois-ci c'est le programme qui doit gérer les rafraîchissements et les actions utilisateur. (voir plus loin à FONCTIONS AVANCÉES pour les détails). Pour donner la main au serveur X-windows, après l'initialisation de l'application, appeler :

```
int startX11();
```

Cette procédure boucle en attendant les entrées clavier ou souris et revient si l'une des fonctions passées en argument par l'utilisateur appelle la procédure :

```
int terminateX11(int n);
```

la valeur "n" (positive ou nulle) est alors rendue en valeur de retour par `startX11`.
Il est possible aussi de tuer la fenêtre sans "ressortir" par la fonction startX11() par la procédure :

```
int closeX11();
```

graphiques :
Les sous-fenêtres graphiques sont des régions rectangulaires de la fenêtre easyX11 composées de deux couches :
– Une couche image, qui peut être vide, remplie entièrement d'une image noir-et-blanc ou remplie entièrement d'une image couleur suivant le mode graphique l'aspect à l'écran des images peut varier(*) (par exemple les images couleurs sont rendues en niveau de gris en mode MONO, et les images noir-et-blanc sont colorées en mode PSEUDO).
– Une couche graphique en couleurs easyX11 (transparent, rouge, vert, bleu, noir ou blanc) au dessus de la couche image. Les points où cette couche a la valeur 0 (transparent), la couche image est visible en dessous.

Pour créer une sous-fenêtre graphique :

```
int newgraphic(int position_gauche, int position_haut, int largeur_en_pixels, int
hauteur_en_pixels, int (*fonction_ou_NULL)(int numero_graphique, int x, int y, int numero_de_bouton));
```

Sa position est relative à la fenêtre ouverte par `openX11()` et ses dimensions sont en pixels. La sous-fenêtre est initialement uniformément noire. La procédure `newgraphic` rend un entier strictement positif qui identifiera la sous-fenêtre pour la procédure interactive "fonction()" fournie par l'utilisateur ainsi que pour les procédures `writegraphic()` `closegraphic()` etc décrites ci-dessous. La procédure passée en argument "fonction()" est appelée quand on clique sur la fenêtre en passant comme paramètres le numéro de la sous-fenêtre, les coordonnées en pixels par rapport au coin haut gauche de la sous-fenêtre graphique, et le numéro de bouton (voir code plus haut).

Pour écrire dans la couche image et/ou dans couche graphique, on peut utiliser les fonctions :

```
int writegraphic(int numero_graphique, int en_couleur, unsigned char *bleu_ou_niveau_gris,
unsigned char *vert_ou_NULL, unsigned char *rouge_ou_NULL, unsigned char *tableau_contours_ou_NULL);
```

et

```
int drawgraphic(int numero_de_graphique, unsigned char *bytes);
```

(*) On peut aussi le faire en après, de même qu'il est possible de changer l'icône par défaut, si le window manager *le permet, mais il faut faire un changement de propriété de la fenêtre (voir le code de listX11 pour un exemple).*
(*) sauf si les pixels correspondant ont été recopiés de la couche graphique par `putgraphic`.

Si bleu_ou_niveau_gris≠NULL et en_couleur=0, la couche image est écrite en noir-et-blanc et les arguments vert_ou_NULL et rouge_ou_NULL sont ignorés. si en_couleur≠0 la couche image est écrite en couleur avec les trois composantes colorées passées en argument. Noter que les tableaux passées en argument doivent faire exactement un caractère par pixel dans la sous-fenêtre. Les pixels sont rangés de gauche à droite et de bas en haut. Noter aussi que `writegraphic(n,0,tableau,NULL,NULL,NULL);` et `writegraphic(n,1,tableau,tableau,tableau,NULL);` ne donnent pas les mêmes résultats sur la plupart des écrans (la qualité obtenue avec la seconde est bien moins bonne en général...)
Le dernier argument s'il est différent de NULL est écrit dans la couche graphique, s'il est NULL et que bleu_ou_niveau_gris est non NULL, la couche graphique est mise à zéro. `drawgraphic(numero_de_graphique,bytes);` est simplement une abréviation pour `writegraphic(numero_de_graphique,?,NULL,?,?,bytes);`.
A l'exception prés que `drawgraphic(n,NULL);` efface la couche graphique alors que `writegraphic(n,?,NULL,?,?,NULL);` n'efface aucune des couches. Ces deux procédures remontent et rafraîchissent la sous-fenêtre même si aucune couche n'est modifiée.

Il est possible d'effacer la couche image et d'initialiser (ou effacer) la couche graphique par la fonction :

```
int cleargraphic(int numero_de_graphique,int couleur);
```

Si couleur=0 (TRANSPARENT) la couche graphique est supprimée. Cette fonction dés-alloue en outre la couche image (effet similaire à celui d'un `free()`).

Il est possible de recopier la couche graphique telle quelle dans la couche image et d'effacer la couche graphique, l'aspect de la fenêtre restant strictement identique. On peut désirer faire ça pour deux raisons principalement :
Pour accélérer le rafraîchissement d'une sous-fenêtre dont la couche graphique ne sera pas modifiée sans la couche image (le rafraîchissement se fera en un passage, au lieu du rafraîchissement de la couche image puis de la couche graphique).
L'autre raison possible est de modifier rapidement un graphisme en respectant un autre graphisme (par exemple déplacer un carré sur un dessin complexe en le traçant en transparent à l'ancienne position puis en blanc à la nouvelle position à chaque mouvement de souris. Si l'on ne fait pas de recopie, le carré effacera le dessin en se déplaçant et il faudra soit "réparer" le dessin complexe entre chaque déplacement de souris (long) soit laisser le dessin se dégrader (laid voire gênant s'il s'agit d'aligner le carré avec des élément du dessin).
La procédure de recopie est :

```
int putgraphic(int numero_de_graphique, unsigned char *bytes);
```

Si le second argument (`bytes`) est `NULL`, la couche graphique avant l'appel est reportée dans la couche image. Noter qu'après cet appel, la couche graphique est vide (complètement à 0).

Il est possible de relire la couche graphique dans un tableau de caractère par la procédure :

```
int readgraphic(int numero_de_graphique, unsigned char *bytes);
```

Les quatre procédures suivantes modifient la couche graphique d'une sous-fenêtre graphique. Toutes quatre ne modifient que la partie de la fenêtre easyX11 correspondant à la sous-fenêtre, par contre elles ne modifient pas l'ordre d'empilement des sous-fenêtres. Ce choix qui peut paraître inélégant (en particulier un trait tracé sur une sous-fenêtre surcharge les boîtes éventuellement placées au dessus) est dicté par des considérations d'efficacité : si ces fonctions remontaient leur sous-fenêtre graphique, une opération de type *clic and drag* (cliquer-traîner) avec affichage des coordonnées sur deux fenêtres se chevauchant entraînerait une alternance de passage devant/derrière des fenêtres à chaque déplacement de la souris. Si vous voulez explicitement amener au premier plan une sous-fenêtre graphique faire `raisegraphic(numero);`. Les trois procédures inscrivent un point, une ligne ou un texte sur la couche graphique :

```
int plotgraphic(int numero_de_graphique, int x, int y, int color);
int linegraphic(int numero_de_graphique, int x1, int y1, int x2, int y2, int couleur, int epaisseur);
int trianglegraphic(int numero_de_graphique, int x1, int y1, int x2, int y2, int x3, int y3, int couleur);
int printgraphic(int numero_de_graphique, int x, int y, char *chaine, int couleur);
```

Pour des raisons d'efficacité des transferts vers le serveur X-windows, la première procédure est tamponnée, c'est-à-dire que les points ne sont pas tracés immédiatement à l'écran, mais par paquets (de

4000 pour chaque couleur) le tampon est vidé explicitement par les deux autres fonctions (qui l'appellent en sous-programme) ou par l'appel de `plotgraphic(-1,?,?,?);`.
Dans toutes ces procédures, les coordonnées sont relatives au coin haut gauche de la sous-fenêtre, les lignes sont tracées dans la direction x2,y2 → x1,y1 et le pinceau utilisé est un carré de coté 2×épaisseur+1 centré sur le point tracé. Les pixels marqués sont identiques d'un écran à l'autre (algorithme du Monopoly) c'est plus lent mais rend les graphiques reproductibles.
Les coordonnées du texte correspondent au coin haut gauche du texte qui apparaîtra (la hauteur de la ligne de base est donnée par la variable eX_basechar). La fonte est la fonte courante, et l'espacement des caractères est proportionnel. La longueur en pixel d'une chaîne de caractère est rendue en valeur de retour par `printgraphic(-1,?,?,chaine,?);`. Noter qu'une chaîne de caractères peut faire plusieurs lignes (le caractère "\n" passe à la ligne, à la verticale du premier caractère de la première ligne) et que la longueur rendue par `printgraphic(-1,...` est celle de la ligne la plus longue composant la chaîne.

Pour supprimer une sous-fenêtre graphique, il suffit d'appeler :

```
int closegraphic(int numero_graphique);
```

boîtes :

Les boîtes sont des sous-fenêtres qui ne contiennent que des caractères, espacés de façon constante et tous de la même couleur (dans une boîte donnée). Si l'on poursuit l'analogie easyX11↔*window manager*, les boîtes sont en quelque sorte les micro-terminaux. Toutes les boîtes présentes dans la fenêtre easyX11 utilisent la même fonte de caractère. En cas de changement de fonte, toutes les boîtes sont retracées avec la nouvelle fonte, et leur dimension est ajustée (le coin haut gauche, le nombre de lignes et de colonnes étant conservé). Pour créer une boîte :

```
int newbox(int x, int y, int largeur_en_caract, int hauteur_en_caract, int video_inverse, int (*fonction_ou_NULL)());
```

La position x,y est celle du coin haut gauche de la boîte par rapport au coin haut gauche de la fenêtre easyX11. Les dimensions en pixels de la boîte dépendent de la fonte courante. Elles sont données par les expressions : largeur_en_caract×eX_largchar+2 et hauteur_en_caract×eX_longchar+2. l'argument video_inverse vaut 0 pour des caractères en blanc sur fond noir, 1 pour des caractères en noir sur fond blanc. Plus généralement video_inverse peut être donné comme un nombre octal de la forme `0nm` (soit 8×n+m) où n est la couleur du fond et m la couleur des caractères. Les codes de couleurs sont ceux donnés plus haut avec 6 pour le gris 50% et 7 qui pour le fond signifie la couleur du fond de fenêtre. Si n=7 en outre, le cadre de la boîte est supprimé. La fonction passée en argument est appelée avec 4 arguments quand on clique sur la boîte : le numéro de boîte, la position x,y en caractères dans la boîte et le numéro de bouton.

Il est possible d'effacer une boîte et de changer éventuellement sa couleur par la fonction :

```
int clearbox(int numero_de_boite, int video_inverse);
```

Si le numero_de_boite est 0, la couleur du fond de la fenêtre easyX11 est changée pour celle donnée dans video_inverse (1 à 6) si video_inverse=0, toute la fenêtre easyX11 est retracée.

Pour écrire du texte dans une boîte :

```
int writebox(int numero_de_boite, char *texte);
```

La chaîne de caractères peut contenir des retours chariot ("\n") et éventuellement déborder de la ligne. Le point d'insertion du texte, qui est initialement (et après clearbox) en haut à gauche se déplace vers le bas. Si la dernière ligne est complètement pleine à l'arrivée d'un caractère, ou qu'un retour chariot est transmis sur la dernière ligne, toutes les lignes sont décalées d'un cran vers le haut. Cette procédure rafraîchit et remonte la boîte, elle est donc relativement lente. Si vous avez de nombreuses lignes à afficher, remplissez une chaîne de caractère de l'ensemble des lignes et appelez writebox une seule fois. `raisebox(n);` peut être utilisé pour remonter une boîte sans modifier son contenu.

Il est possible de faire entrer ou modifier une chaîne de caractère au clavier par l'utilisateur dans une boîte. La procédure suivante est un sorte de micro-éditeur. L'édition d'une chaîne de caractère est limitée au nombre de caractères visibles dans la boîte et ne doit pas contenir de retours chariot. La chaîne de caractère est considérée comme une seule ligne éventuellement découpée pour l'affichage en plusieurs lignes physiques. Pour éditer une chaîne, la passer en argument à :

```
int readbox(int numero_de_boite, char *texte_initial_a_editer);
```

ATTENTION : La chaîne de caractère doit nécessairement être initialisée (au pire à "") sinon, la procédure affichera les caractères contenus à l'adresse passée en argument jusqu'au premier caractère 0 (ou la première `segmentation fault` !). La chaîne de caractère doit aussi être assez grande pour contenir le texte entré. En particulier il n'y a pas de vérification de la longueur de la chaîne entrée ! Si la chaîne de caractère initiale contient des retours chariot, il est impossible d'éditer la portion située avant le dernier retour chariot et de remonter avant le premier caractère affiché dans la boîte. Dans la description qui suit, le "début de la chaîne" désigne le premier caractère modifiable, c'est-à-dire le plus en avant de (1) le premier caractère de la chaîne initiale, (2) le caractère suivant le dernier retour chariot, s'il y en a et (3) le premier caractère affiché au coin haut gauche de la boîte. Les touches utilisables pour l'édition sont les suivantes :

[delete]	efface la partie éditable (jusqu'au début de la chaîne)
[Home]	positionne le curseur en début de chaîne
[End]	positionne le curseur en fin de partie éditable
[Backspace]	efface le caractère à droite du curseur
[Return]	termine l'entrée de texte

Les autres caractères sont insérés à la position courante du curseur. Il est possible pendant cette opération d'utiliser le mécanisme de sélection (copier-coller) de et vers une boîte easyX11 (la même, une autre de cette fenêtre ou d'une autre fenêtre(*)

Pour supprimer un boîte il suffit d'appeler :

```
int closebox(int numero_de_boite);
```

FONCTIONS AVANCÉES

curseurs :

Il est possible de changer le curseur (ou le pointeur) qui matérialise la position de la souris. Deux fonctions sont utilisables, la première d'un usage simple donne le choix entre cinq curseurs standards de X-windows en noir sur fond blanc. La seconde permet de créer un curseur personnalisé, éventuellement en couleur :

```
int changecursor(int cursor_no);
```

donne pour 0 le curseur en croix + (XC_crosshair), pour 1 le curseur d'attente (XC_watch), pour 2 un curseur en flèche vers le haut (XC_center_ptr), pour 3 une main pointée (XC_hand2). −1 donne le curseur par défaut de la fenêtre (dépend du *window manager*) et 4 remet le curseur personnel (s'il a été défini).

Pour définir et installer un curseur personnel, utiliser la procédure :

```
int definecursor(int num, char *foreground, char *background, int foregroundcolor, int backgroundcolor);
```

suivant les arguments utilisés ou laissés à NULL, le résultat est différent :
(n,NULL,NULL,foregroundcolor,backgroundcolor) crée et installe un curseur le la fonte curseur standard de X-windows dans la couleur spécifiée (voir à la fin de la documentation Xlib pour la liste et les valeurs n correspondant aux curseurs standards. (n,fonte,NULL,foregroundcolor,backgroundcolor) crée et installe un curseur d'une fonte spécifiée, c'est-à-dire utilise le caractère n comme trait et le caractère n+1 comme masque. Il y a des fontes de curseurs dans certains systèmes comme Openwin sur Sun. (?,bitmap,maskbitmap,foregroundcolor,backgroundcolor) crée et installe un curseur à partir de deux bitmaps X-windows. Les couleurs sont codées de la façon standard 0=invisible(fond)/gris(curseur) 1=bleu 2=vert 3=rouge 4=blanc 5=noir 6=gris, à l'exception du cas où les deux couleurs sont 0 : le curseur est alors effacé. Si les arguments fonte,bitmap ou maskbitmap ne commencent pas par "/" le *directory* est celui donné par la variable d'environnement `EASYX11DIR`.

(Si l'autre fenêtre tourne sur une autre unité centrale, il vaut mieux effectuer assez lentement les opérations de copier coller, car en cas de charge importante du système, la fenêtre qui émet peut ne pas notifier assez rapidement la prise de sélection. La demande de copie peut arriver trop vite dans l'autre fenêtre provoquant un échec de transmission donc la copie à partir du cutbuffer de la sélection précédente.)*

fontes et sélection :

On peut changer la fonte utilisée dans la fenêtre easyX11 par :

```
int changefont(char *filename);
```

Le nom de fichier doit être celui d'une fonte easyX11, et si le nom ne commence pas par "/", le fichier est pris dans le *directory* donnée par la variable d'environnement `EASYX11DIR`. Noter que cette fonction provoque le changement des caractères affichés dans les boîtes (mais ne modifie pas, bien sûr, les caractères tracés précédemment dans une sous-fenêtre graphique par printgraphic !).

On peut changer en marche le mode de colorisation (défini normalement à l'appel de `openX11` par le fichier de configuration dont le chemin est donné par la variable d'environnement `EASYX11FILE`). Pour cela passer le nouveau mode à la fonction :

```
int changemode(char *mode);
```

le mode doit être donné en majuscule. Il faut ensuite ré–afficher toutes les images (par `writegraphic`) dans les sous-fenêtres graphiques. (les contours et les boîtes sont rafraîchis automatiquement).

Si une boîte n'a pas de fonction, elle est utilisable même en dehors de l'exécution de readbox pour les opérations de copier. En insérant, la fonction suivante en sixième argument de newbox :

```
int textselect(int numero_de_boite, int x, int y, int numero_de_bouton);
```

cette fonction rend 0 si le numéro de bouton était différent de 1, –1, –3 ou 3 (ce qui rend possible l'utilisation en suite du bouton du centre et du bouton de droite avec Shift ou Ctrl) 1 si la sélection à été effectuée et –1 si elle a échouée. Si l'on veut pouvoir sélectionner et utiliser la boîte simultanément, le source de la fonction passée en argument à newbox doit ressembler à ceci :

```
int ma_fonction_de_boite (numero_de_boite, x, y, numero_de_bouton)
int numero_de_boite, x, y, numero_de_bouton;
{
if ( textselect(numero_de_boite, x, y, numero_de_bouton))
return(0);
```

...suite de ma_fonction_de_boite qui peut utiliser les valeurs ±2, ±4, ±5, ±6 ou ±7 de `numero_de_bouton`*...*

```
}
```

interrogation du serveur :

Il est possible de capturer les mouvements de la souris et d'appeler une fonction définie par le programmeur à chaque mouvement de la souris. Il est aussi possible par la même fonction de faire qu'une seule sous-fenêtre graphique reçoive les clics de souris. Pour cela il faut appeler :

```
int querygraphic(int numero_de_graphique, int (*fonction_ou_NULL)(int x, int y));
```

Si la fonction est NULL, la souris est libérée, tous les mouvements non encore traités sont perdus et les clics sont envoyés à la sous-fenêtre sur laquelle ils sont faits. Sinon, la fonction est appelée à chaque déplacement de la souris avec des coordonnées relatives à la sous-fenêtre graphique de numéro spécifié. Si ce numéro est 0, les coordonnées seront relatives au coin haut-gauche de la fenêtre easyX11. Si le numéro est l'opposé d'un numéro de sous-fenêtre graphique, les clics ultérieurs seront re-dirigés sur cette sous-fenêtre (c'est-à-dire que ce sera la fonction passée au newbox qui l'a crée qui sera appelée, avec une position x,y éventuellement en dehors de la sous-fenêtre, voire même en dehors de la fenêtre.

Il est possible de demander à l'interface les positions des sous-fenêtres par les deux fonctions :

```
int graphicposition(int numero_de_graphique, int *position_gauche, int *position_haut);
```

et

```
int boxposition(int numero_de_boite, int *position_gauche, int *position_haut, int
*largeur_en_pixels, int *hauteur_en_pixels);
```

Les arguments non utilisés peuvent être mis à NULL. La seconde fonction rend la hauteur et la largeur, car elle dépend de la fonte de caractère en cours. Si le numéro de boîte est 0 dans la seconde fonction, les paramètres retournés sont ceux de la fenêtre easyX11.

On peut aussi déplacer une sous-fenêtre sans altérer son contenu par les deux fonctions :

```
int movegraphic(int numero_de_graphique, int position_gauche, int position_haut);
```

et

```
int movebox(int numero_de_boite, int position_gauche, int position_haut);
```

De même l'ordre d'empilement des sous-fenêtre peut être modifié par les quatre fonctions suivantes :

```
int raisegraphic(int numero_de_graphique);
```

```
int lowergraphic(int numero_de_graphique);

int raisebox(int numero_de_boite);

int lowerbox(int numero_de_boite);
```

Il est aussi possible d'interroger le serveur sur la position du curseur et en particulier sur quelle sous-fenêtre recevrait un clic si un bouton de la souris était manipulé. Cette fonction est:

```
int cursorposition(int *type, int *numero, int *x, int *y);
```

qui retourne 0 si le curseur est dans la fenêtre easyX11. Les champs suivants (pouvant être mis à NULL si non utilisés) décrivent respectivement: Le type de sous-fenêtre (−2 = hors fenêtre, −1 = fond de fenêtre, 0 = sous-fenêtre graphique et 1 = boîte). Le numéro de sous-fenêtre graphique ou de boîte et la position dans la sous-fenêtre.

Il est enfin possible de vérifier s'il y a des événements X-windows en attente de traitement. Cette fonction peut être utilisée pour pouvoir interrompre en cours de calcul certaines opérations très calculatoires, ou pour faire un programme non asynchrone qui utilise easyX11. Pour cela il faut invoquer une première fois startX11 pour activer easyX11, puis dans une fonction appelée (par exemple key_entry ou une fonction de boîte) lancer les calculs. Tous appel de

```
startX11();
```

dans ce contexte rend un entier qui signifie par ordre de priorité décroissante :

code	signification
1	pression de touche au clavier
2	pression de bouton souris
3	relâche d'un bouton souris
4	déplacement de la souris
5	changement de taille ou d'empilement concernant la fenêtre
6	changement de sélection ou demande de sélection
7	franchissement de la frontière de la fenêtre par le curseur ou changement de colormap
0	rien de tout ça

startX11 utilisé dans ce contexte ne permet pas d'ignorer l'événement ou de le traiter sans sortir de la procédure courante (c'est-à-dire sans rendre la main à la première instance de startX11). Il y a donc une fonction plus détaillée pour ce faire. (je rappelle que l'utilisation de cette fonction est contraire à la philosophie X11...)

```
int eventX11(int masque, int action);
```

qui rend le même code que startX11 pour les événements acceptables, où masque est obtenu en additionnant 2 à la puissance des codes acceptés (p.ex. si l'on veut les pressions de touches et les déplacements de la souris, masque = $2^1 + 2^4 = 18$) et action détermine ce que l'on en fait (0=on le laisse pour après le retour, 1=on l'oublie, −1=on le traite tout de suite, 2 ou −2=comme 1 ou −1 mais bloque si aucun événement jusqu'au prochain événement compatible). Attention: si vous utilisez action = −1 pour une pression de bouton, et que la souris est sur une boîte ou un graphique, la fonction passée en argument à newbox/newgraphic sera exécutée avant que eventX11 ne retourne le code.

variables internes de easyX11 :

Les déclarations suivantes permettent l'accès direct à la mécanique de easyX11. Sachez si vous tenez à les utiliser (1) que ce n'est pas toujours évident (2) que rien ne garantit que les structures restent les mêmes dans les prochaines versions de easyX11...

Pour les utiliser, il faut déclarer la constante symbolique `_EASYX11_INTERN_` avant `#include "easyX11.h"` dans votre source.

`int (*eX_f_motion)();` fonction appelée aux déplacement de la souris

`int eX_base_motion;` fenêtre par rapport a laquelle on querygraphic

`#include <X11/Xlib.h>` déclarations des fonctions de Xlib

`extern Display *eX_display;` terminal, fenêtre, contexte graphique et écran de easyX11

`extern Window eX_win;`

`extern GC eX_gc;`

`extern int eX_screen;`

`extern int eX_BLACK,eX_WHITE,eX_BLUE,eX_GREEN,eX_RED,eX_GREY;` couleurs (numéro dans le *colormap*) correspondant aux couleurs easyX11

```
extern struct graphictype
 {
 int active,activecontours,lastrefresh;
 int x,y,width,height;
 XImage *image;
 union {
  int *data;
  unsigned char *byte;
  }pixel;
 unsigned char *contours;
 unsigned char lut[256];
 int (*function)();
 } *eX_graphic;
```
descripteur des sous-fenêtres graphiques
```
extern struct boxtype
 {
 int active,lastrefresh;
 int x,y,width,height,inversevideo,cursor;
 unsigned char *string;
 int (*function)();
 } *eX_box;
```
descripteur des boîtes

Attention : eX_graphic n'est alloué qu'à l'ouverture de la première sous-fenêtre graphique, et eX_box n'est alloué qu'à l'ouverture de la première boîte.

TOOLKIT :

Les fonctions suivantes ne sont pas réellement des fonctions easyX11, mais un emballage de traces sur des sous-fenêtres graphiques. En particulier, les *id* rendus par `newlift`, `newdial`, `newpopup`, `selectfile`, `newviewtext` et `newviewimg` sont des numéros de graphiques. Attention toutefois, `newpopup` crée deux sous-fenêtres, une pour l'ombre et une pour le menu, de même `selectfile`, `newviewtext` et `newviewimg` peuvent ouvrir des ascenseurs.

Ascenseurs :

Pour les ascenseurs/thermomètres l'interface est la suivante (compatible avec le fonctionnement de l'ascenseur de Xterm) :

CRÉATION D'UN ASCENSEUR :

```
int newlift(int x, int y, int lar, int hau, int horiz, int longueur, int epaisseur, int couleur,int f_resh(), int f_fastresh());
```

avec les paramètres suivants :

x position du bord gauche de l'ascenseur
y position du bord haut de l'ascenseur
lar largeur de l'ascenseur (échelle comprise si vertical)
hau hauteur de l'ascenseur (échelle comprise si horizontal)
horiz 1 si ascenseur horizontal, 0 si vertical
longueur longueur du pave mobile de l'ascenseur (si ≤0 l'ascenseur est du type thermomètre)
epaisseur largeur de la colonne (hauteur si horizontal) peut être < lar si on prévoit une échelle (si positif à partir du coté gauche/haut si négatif à partir du coté droit/bas)
couleur couleur de l'ascenseur, même définition que pour les "box"s easyX11
f_resh fonction appelée quand l'ascenseur a fini de se déplacer
f_fastresh fonction appelée pendant le déplacement de l'ascenseur

Telle qu'est prévue l'interface, si l'on clique avec le bouton de gauche, l'ascenseur est déplacé de sa longueur (en valeur absolue) vers le bas ou la gauche, c'est-à-dire vers les valeurs positives de la position. f_resh(id,i) est appelée quand on relâche le bouton avec id le numéro de l'ascenseur et i valeur entière de la nouvelle position de l'ascenseur.

Si l'on clique avec le bouton de droite, l'ascenseur est déplacé de sa longueur vers le haut ou la droite. f_resh(id,i) est appelée quand on relâche le bouton.

Si l'on enfonce le bouton du centre, la souris est capturée, l'ascenseur suit ses déplacements, et la fonction f_fastresh(id,i) est appelée successivement avec id le numéro de l'ascenseur et i les positions de

l'ascenseur. Quand on relâche le bouton, l'ascenseur s'immobilise et la fonction f_resh(id,i) est appelée avec la dernière position de l'ascenseur.

ÉCRITURE D'UNE GRADUATION :

```
int printlift(int id, int couleur, int pos, int len, int x, int y, char *texte);
```

avec les arguments :

id le numéro d'ascenseur

couleur code couleur easyX11 de la graduation

pos la position du trait de graduation

len la longueur du trait de graduation (si 0 pas de graduation)

x,y la position (relative à l'origine de la graduation) du texte. Si =0 centrage du texte du coté opposé à la graduation.

texte une chaîne de caractères à imprimer à coté de la graduation (si NULL pas de texte)

RE-CONFIGURATION DE L'ASCENSEUR :

Pour changer la position de l'ascenseur, sa couleur, effacer l'échelle, utiliser la fonction :

```
int clearlift(int id, int pos, int len, int couleur, int efface_echelle);
```

avec les paramètres :

id le numéro d'ascenseur

pos la nouvelle position du pavé de l'ascenseur (−1 si pas de changement)

len la nouvelle longueur du pavé (−1 si pas de changement)

couleur nouvelle couleur de l'ascenseur (−1 si pas de changement)

efface_echelle 0=ne pas effacer le reste de la fenêtre, 1=effacer toute la fenêtre

FERMETURE D'UN ASCENSEUR :

```
int closelift(int id);
```

Cadrans :

Pour l'interface d'un cadran, l'interface est très similaire (par contre elle est différente de celle du *dialbox* Stardent) :

CRÉATION D'UN CADRAN :

```
int newdial(int x, int y, int lar, int lon, int x0, int y0, int r, int ang_min, int ang_max, int d_ang,
int r_aig, int lar_aig, int r_axe,couleur, int f_resh(), int f_fastresh());
```

avec les paramètres :

x,y position de l'axe du cadran

lar,lon dimension de la sous-fenêtre contenant le cadran

x0,y0 axe du cadran par rapport au coin haut gauche de la sous-fenêtre

r rayon total du cadran

ang_min,ang_maxplage angulaire du cadran en degrés (si ang_min=ang_max=0 cadran rotatif cyclique sur 0→359 degrés)

d_angvariation angulaire correspondant aux boutons gauche et droite en degrés si ≤0 le cadran ne peut être déplacé à la souris

r_aig rayon de la pointe de l'aiguille à partir de l'axe

lar_aig largeur de la base de l'aiguille au niveau de l'axe en degrés de part et d'autre

r_axe rayon du pivot de l'aiguille

couleur couleur du cadran, même code que pour les "box"s easyX11

f_resh fonction appelée après le déplacement de l'aiguille

f_fastresh fonction appelée pendant les déplacements de l'aiguille avec le bouton du centre

Comme pour l'ascenseur, le bouton de gauche augmente l'angle, celui de droite le diminue, et celui du centre asservit l'aiguille aux déplacements de la souris.

ÉCRITURE D'UNE GRADUATION :

```
int printdial(int id, int couleur, int pos, int r, int len, int dr, int dpos, char *texte);
```

avec les arguments :

id numéro du cadran

couleur couleur de la graduation (code de couleur de contours easyX11)

pos position angulaire de la graduation

r rayon de départ de la graduation

len longueur de la graduation
dr,dpos position du texte par rapport au bout de la graduation. Si len>0 (*resp.* len<0) et dr=dpos=0, le texte est cadre à l'intérieur (*resp.* extérieur) du cercle de rayon r+len
texte légende de la graduation (ou NULL)

RE-CONFIGURATION DU CADRAN :
```
int cleardial(int id, int pos, int couleur, int efface_echelle);
```
avec les arguments :
id numéro du cadran
pos nouvelle position de l'aiguille
couleur nouvelle couleur du cadran (comme pour les "box"s easyX11) si −1 pas de changement
efface_echelle région à retracer ou effacer : 0=l'axe seulement, 1=l'axe et le disque du cadran, 2=toute la fenêtre)

FERMETURE D'UN CADRAN :
```
int closedial(int id);
```

menus popups :

Les menus transitoires, (*popup*) ont un aspect proche de celui de twm en deux sous-fenêtres graphiques, une pour l'ombre et une pour le menu. Les sous-menus sont invoqués quand la souris est déplacée sur le tiers droite de la ligne ou quand on relâche le bouton sur la ligne. Le menu principal reste jusqu'à un relâcher de bouton (dans le menu, un sous-menu ou hors du menu).
INVOCATION D'UN MENU :
```
int newpopup(int x, int y, int ouvert, char **titres, int *execut, int couleur0, int couleur1, int couleur2, int function());
```
avec les arguments :
x,y position du menu
ouvert souris capturée et amenée sur la première ligne (si =1) souris libre (si =0)
titres (char **) liste des intitulés des lignes du menu terminée par NULL
types (int *) types d'entrées 0=non exécutable (titre) 1=exécuté si le bouton est relâché, 2=exécuté quand le pointeur est amené sur le 1/3 droite (sous-menus), 3=équivalent à 1 et 2 simultanément et ≥4=non exécutable.
couleur0 couleur des lignes de type 0
couleur1 couleur des lignes de type 1 à 3
couleur2 couleur des lignes de type 4 et plus
fonction fonction appelée avec le numéro de menu et le numéro de ligne (−2 si l'on sort, −1 si l'on clique en dehors)

RÉACTIVATION D'UN MENU :
```
int openpopup(int id, int pos);
```
réactive le menu id en positionnant le curseur sur la ligne pos. Sert à retourner dans un sous-menu.

FERMETURE D'UN MENU :
```
int closepopup(int id);
```
Pour obtenir des menus au format twm, il faut que le menu principal ignore le code −2 et fasse un closepopup sur −1, les sous-menus par contre appellent closepopup sur −1 et −2 avec en outre appel à openpopup(parent_id,ligne_du_sous_menu).

Sélecteur de fichier :

Le sélecteur de fichier permet de naviguer dans le système de fichiers et d'y sélectionner un fichier. Cet utilitaire s'appelle par :
```
int selectfile(int x, int y, int larg, int haut, FILE *tmp, char *dir, char *name, int bg_color, int fg_color, int (*fonction)(int n_select));
```

Où x, y, larg & haut déterminent une région (approximative) en pixels où sera positionné la liste de fichiers. Si le directory contient trop de fichiers, un ascenseur vertical permet de dérouler la sous-fenêtre. Si les lignes sont trop longues pour la sous-fenêtre, un ascenseur horizontal permet de se déplacer sur la ligne. Les lignes (,,,) en début ou fin de fenêtre permettent de sauter de page en page en cliquant dessus (elles disparaissent si l'on est en début ou fin de liste resp.
Le fichier tmp est un fichier temporaire utilisé pour stocker les lignes de la liste, il doit impérativement être ouvert en mode w+ (lecture et écriture).
dir contient en entrée le nom du *directory* de départ. file contient en entrée un gabarit de sélection (*pattern*). Les noms de fichier (mais pas ceux des *directories*) doivent se conformer au gabarit pour être listés. Un gabarit est une chaîne de caractères dont les caractères doivent correspondre à ceux des noms de fichier à l'exception des caractères suivants :

?	correspond avec n'importe quel caractère.
\?	correspond au caractère ? lui-même
*	correspond avec n'importe nombre de caractères (même zéro).
*	correspond au caractère * lui-même
[...]	correspond avec l'un des caractères listées entre les crochets
\[	correspond au caractère [lui-même

Dans les listes de caractères, le caractère - permet de définir des intervalles de caractères (p.ex. [0-9] correspond à un chiffre) le caractère] lui même doit être le premier de la liste si on veut inclure le caractère] dans la liste(*) et le caractère - doit être le premier ensuite si on le veut dans la liste.
Pendant la sélection, il est possible de modifier le gabarit ou le *directory* du clavier en cliquant sur la première ligne de la liste (qui affiche le *directory* entre crochets et le gabarit).
En fin de sélection, dir et name contiennent respectivement le nom du *directory* et celui du fichier, et la fonction est appelée avec l'argument id (l'identificateur rendu par selectfile). Attention, easyX11 ne vérifie pas qu'il y a de la place dans les chaînes de caractères dir et name...
bg_color et fg_color sont respectivement les couleurs du fond et des caractères.

Il est possible d'interrompre (p.ex pour une touche "cancel") une sélection autrement qu'en sélectionnant un fichier, il suffit d'appeler la fonction :
`int` **abortselectfile**`(int id);`
Où id est l'identificateur rendu par selectfile.

Visualisation de texte :

Cet outil permet de visualiser un fichier texte avec éventuellement des ascenseurs si le texte dépasse du cadre de l'outil. Il se crée par :
`int` **newviewtext**`(int x, int y, int larg, int long, FILE *text, int bg_color, int fg_color, int hg_color, int (*fonction)(int id, int col, int lig, int n_bouton));`
Comme précédemment, x, y, larg & long définissent la position et la dimension en pixels de l'outil. Le fichier "text" contient le texte à visualiser. bg_color est la couleur du fond et fg_color celle des caractères. hg_color est la couleur des caractères de code >128. Si elle est égale à fg_color, ces caractères sont affichés tels quels (caractères accentués en général). Si elle est différente, le caractère de code–128 est affiché dans la couleur hg_color. Cela permet de coder la "sur-brillance" dans une chaîne de caractères ordinaire.
La fonction passée en argument est appelée chaque fois que l'on clique (ou que l'on relâche le bouton de la souris) sur un caractère du texte. Les arguments de la fonction sont : l'identificateur de l'outil (rendu par `newviewtext`), la position (colonne et ligne) dans le texte du caractère et le numéro du bouton enfoncé ou relâché.
Comme pour le sélecteur de fichier, la première ligne affichée peut être précédée (et la dernière ligne affichée suivie) d'une ligne (,,,) qui permet de sauter de page en page dans le texte si possible.
Le texte est affiché initialement à partir de la position courante dans le fichier passé en argument. Si le fichier est modifié pendant l'exécution de newviewtext(), l'affichage devient incorrect. On peut toutefois accepter des modifications du fichier grâce à la fonction :
`int` **refreshviewtext**`(int id, int col_gauche, int lig_haut);`

(*) attention, ceci est légèrement différent de ce que l'on obtient sous shell *par exemple, en argument d'un commande* `ls`, `[]` *ne correspond avec aucun caractère et* `[-ab]` *ne correspond qu'avec les caractères* `a` *et* `b`.

où id est l'indentificateur rendu par newviewtext(). Si les longueurs des lignes ne sont pas modifiées on peut passer –id à la fonction pour l'accélérer. col_gauche ou lig_haut peuvent être –1 si l'on ne désire pas les changer.

`int` **viewtextposition**`(int id, int *col_gauche, int *lig_haut);`
rend la position dans le texte du caractère situé au coin haut gauche de la partie visible du texte. l'un ou l'autre des pointeurs col_gauche et lig_haut peut être NULL si la quantité correspondante ne nous intéresse pas.

Enfin pour fermer l'outil de visualisation de fichier texte, appeler :
`int` **closeviewtext**`(int id);`

visualisation d'image :

Cet outil permet de visualiser une image dans un cadre (en général, et c'est son intérêt) plus petit que l'image. Si l'image est plus grande que le cadre, des ascenseurs permettent de déplacer la partie affichée. Cet outil s'appelle par :
`int` **newviewimg**`(int x, int y, int larg, int haut, int flag, void *bleu, void *vert, void *rouge, void *contours, int larg_image, int haut_image, int bg_color, int fg_color, int (*fonction)(int id, int x, int y, int bouton));`
Où x, y, larg & long délimitent la région occupée par l'outil, flag indique si l'image est passée en type `FILE*` (si flag≠0) ou en type `unsigned char*` (si flag=0). Bleu, vert & rouge pointent vers les fichiers ou les arrays contenant les composantes de l'image. Si seul bleu≠NULL, l'image est monochrome. Si bleu≠NULL & vert≠NULL l'image est bi-chromatique (canaux affichés magenta et vert). Contours pointe vers le fichier ou l'array contenant la couche graphique (qui peut être NULL bien sûr). De même bleu=NULL et contours≠NULL permet de n'afficher que des graphismes.
larg_image & long_image sont les dimensions de l'image et des contours à afficher. bg_color et fg_color sont les couleurs de fond et de trait des éventuels ascenseurs. La fonction passée en argument est appelée avec comme arguments l'identificateur d'outil, les coordonnées dans l'image et le numéro de bouton, chaque fois que l'on enfonce ou relâche un bouton de la souris sur la partie affichée de l'image.

La fonction :
`int` **moveviewimg**`(int id, int x, int y);`
permet de déplacer par programme la région visualisée par l'outil d'identificateur id. (comme toujours id est la valeur rendue par newviewimg()). Les coordonnés x et y correspondent au coin haut gauche de la partie affichée (pas si l'on est trop bas ou trop à droite cependant...). Si x ou y sont négatifs, la valeur de la coordonnée correspondante n'est pas changée. Si x et y sont négatifs, l'image est rafraîchie (en particulier si le contenu a changé).

`int` **viewimgposition**`(int id, int *x_rel, int *y_rel);`
rend la position dans l'image du coin haut gauche de la partie visualisée de l'image. L'une ou l'autre des variables x_rel ou y_rel peut être NULL si la quantité correspondante ne nous intéresse pas.

On peut tracer sur une visualisation d'image à condition : (1) que l'image soit de type `unsigned char*` (2) que des contours aient été passés à la création de l'outil. Les quatre fonctions suivantes sont les analogues de `plotgraphic`, `linegraphic`, `trianglegraphic` et `printgraphic` sauf que leur coordonnées se rapportent à l'image et non à la sous-fenêtre :
`int` **plotviewimg**`(int numero_de_graphique, int x, int y, int color);`
`int` **lineviewimg**`(int numero_de_graphique, int x1, int y1, int x2, int y2, int couleur, int epaisseur);`
`int` **triangleviewimg**`(int numero_de_graphique, int x1, int y1, int x2, int y2, int x3, int y3, int couleur);`
`int` **printviewimg**`(int numero_de_graphique, int x, int y, char *chaine, int couleur);`
Ces fonctions écrivant directement dans le tableau d'`unsigned char` passé en argument à `newviewimg`, il n'y a pas de fonction "readviewimg". Il est par contre possible de tracer des graphismes directement sur la partie visualisée par les fonctions plot/line/triangle/printgraphic (mais ils seront perdus si l'on se déplace par les ascenseurs ou `moveviewimg`). On peut aussi "fixer" les graphismes par `readgraphic+putgraphic` pour les fonctions interactives (mais il faut le refaire après chaque déplacement de fenêtre).

Enfin l'outil se referme par :
`int` **`closeviewimg`**`(int id);`
où id est l'identificateur de l'outil.

Lecture d'image comprimées ou 16 bits :

Ces quelques fonctions permettent de lire facilement les images des formats TIM non triviaux, il s'agit des formats comprimés 8 et 16 bits ainsi que le format 16 bit. (pour rappel le format "float" est toléré mais pour des applications locales car il n'est généralement pas portable d'une machine à une autre ! Il n'y a pas de fonctions spécifiques d'écriture directe dans les formats comprimés car cela est contraire à la philosophie de easyX11 (ce n'est pas véritablement *easy*). Par contre, on peut avoir à visualiser des images archivées dans une application sans avoir à la décomprimer. Il faut noter toutefois que, sauf pour le format 8 bits, on aura nécessairement à gérer le re-étalement de la dynamique sur l'intervalle visualisable 0 à 255...
Un fichier comprimé 8 bits s'ouvre par la commande :
`char *`**`c8fopen`**`(char *nom_fichier ,int largeur ,int longueur);`
le pointeur rendu est en fait un pointeur vers une grosse structure de décodage. Le fichier ouvert peut être lu ligne par ligne par la commande :
`c8fread``(char *pointeur ,unsigned char *ligne_de_pixels);`
On peut revenir à la première ligne par :
`c8rewind``(char *pointeur);`
Et le fichier se referme par :
`c8fclose``(char *pointeur);`

D'une façon similaire les fichiers comprimés 16 bits sont lus par les commandes :
`char *`**`c16fopen`**`(char *nom_fichier ,int largeur ,int longueur);`
`c16fread``(char *pointeur ,unsigned short *ligne_de_pixels);`
`c16rewind``(char *pointeur);`
`c16fclose``(char *pointeur);`
et les fichiers non comprimés 16 bits (qui s'ouvrent, se rembobinent et se referment normalement par `fopen`, `rewind` et `fclose`) peuvent se lire par :
`u16fread``(FILE* pointeur_fichier ,int nombre ,unsigned short *tableau_de_pixels);`
L'utilisation de cette fonction est recommandé car elle lit deux octets dans le bon ordre (poids fort puis poids faible) alors qu'une commande comme : `fread(unsigned short *tableau_de_pixels ,sizeof(unsigned short),int nombre ,FILE* pointeur_fichier);` FAUX!
ne marche pas sur HewlettPackard (de sexe «poids faible d'abord») ni sur Cray (où les `short` font 64 bits)...