Programmation X Window

Intinsics/Motif (Xt/Xm)

Alexis Nédélec

Ecole Nationale d'Ingénieurs de Brest
Technopôle Brest-Iroise, Site de la Pointe du Diable
CP 15 29608 BREST Cedex (FRANCE)
e-mail : nedelec@enib.fr

Table des Matières

Introduction

Programmation événementielle **client/serveur** de présentation

- ▷ **Serveur X** : gestion d'une file d'événements
  - ◇ quel type d' événements (event-driven)
  - ◇ à quel endroit (dispatch-event)
- ▷ **Client** : applications (bibliothèques Xlib, Xt, Xm)

Gestion de la file d'évènements

- ▷ boucle infinie / switch sur événements
- ▷ gestion de boucle difficile si nombre de fenêtres important

Boîte à outils (Toolkit)

- ▷ bibliothèque **Intrinsics** (Xt), pour la gestion d'évènements

Composants d'interface

- ▷ bibliothèque **Motif** (Xm), ensemble de "Widget" (**Wi**n**d**ow gad**get**)

Etapes de Programmation

Couche de programmation **Intrinsics**

- ▷ création, gestion de fenêtres
- ▷ gestion d' événements X
- ▷ gestion d'actions sur une fenêtre

Cinq étapes fondamentales de programmation

1. Initialisation de la couche Intrinsics
2. Création des widgets de l'application
3. Gestion des événements / réflexes
4. Visualisation de widgets
5. Scrutation de la boucle d'événements

Etapes de Programmation

Sur ce modèle de programmation, fonctions **Xt** fondamentales à utiliser

1. `XtInitialize()`
2. `XtCreateWidget()`
3. `XtAddEventHandler()`,`XtAddCallback()`
4. `XtRealizeWidget()`
5. `XtMainLoop()`

Etapes de Programmation

`XtInitialize()`
- ▷ initialisation de la couche Intrinsics
- ▷ connexion au serveur X
- ▷ allocation des ressources de l'application

`XtCreateWidget()`
- ▷ construction de l'arbre de widgets

`XtAddEventHandler(), XtAddCallback()`
- ▷ gestion de la file d' événements du serveur X
- ▷ gestion de réflexes (callbacks) associés à un widget

`XtRealizeWidget()`
- ▷ création de la fenêtre X utilisée par l'application

`XtMainLoop()`
- ▷ entrée dans la boucle d'événements X

Création d'Interfaces

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>

int
main( int argc , char* argv[] ) {
  Widget    toplevel,msg_widget;
  Arg       wargs[3];
  int       n;
  XmString  msg;

  toplevel=XtInitialize( argv[0], "Debut", NULL, 0, &argc, argv );

  n=0;
  XtSetArg(wargs[n], XmNwidth, 200); n++;
  XtSetArg(wargs[n], XmNheight, 50); n++;
```

Création d'Interfaces

```
  msg=XmStringCreate( argv[1], XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[n], XmNlabelString, msg); n++;

  msg_widget=XtCreateManagedWidget( "msg",
                                    xmLabelWidgetClass,
                                    toplevel,
                                    wargs,
                                    n );
  XtRealizeWidget(toplevel);
  XtMainLoop();
  return 0;
}
```

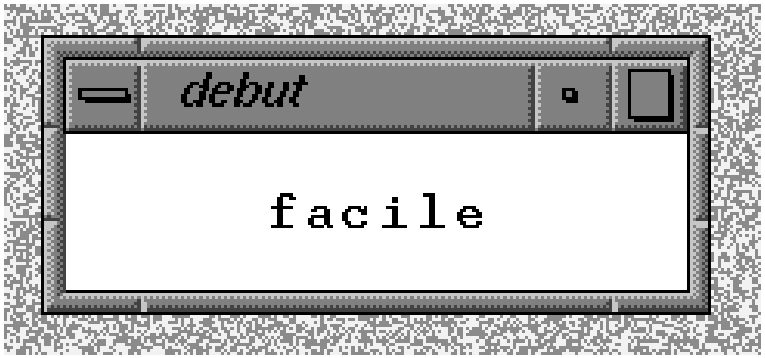
Création d'Interfaces

But de ce programme : affichage de texte dans un widget de type "label"

Compilation de l'application : `$cc debut.c -lXm -lXt -lX11 -o debut`

Exécution de l'application : `$ debut facile`

Affichage



Création d'Interfaces

Fichiers en-tête indispensables

- ▷ `<X11/Intrinsic.h>` : pour toute application Intrinsics
- ▷ `<X11/StringDefs.h>` : pour les chaînes de caractères
- ▷ `<Xm/Xm.h>` : pour toute application Motif
- ▷ `<Xm/Label.h>` : pour les widgets crées par l' application

Ces fichiers doivent être déclarés **dans cet ordre**

Initialisation

```
Widget XtInitialize( String name, String class_name,
                     XrmOptionDescRec* options, Cardinal noptions,
                     Cardinal *argc, char* argv[] )
```

Rôle

- ▷ établir la connexion au serveur X
- ▷ initialiser les ressources de l'application
- ▷ reconnaître les arguments de la ligne de commande
- ▷ rajouter ces arguments en tant que ressources
- ▷ créer dynamiquement le widget racine de l'application

Utilisation

```
toplevel=XtInitialize( argv[0], "Debut", NULL, 0, &argc, argv );
```

Initialisation

Arguments

- ▷ `name` : Nom de l'application (`argv[0]`)
- ▷ `class_name` : Nom de classe d'application
- ▷ `options` : Tableau d'arguments de la ligne de commandes
- ▷ `noptions` : nombre d'options
- ▷ `argc, argv` : idem langage C

Convention Intrinsics / Motif (gestion de ressources)

- ▷ nom d' Application : `"debut"`
- ▷ nom de Classe d'application : `"Debut"`

Initialisation

Initialisation de ressources sur la ligne de commandes

▷ Structure `XrmOptionDesRec` : Gestionnaire de ressources (`Xrm...`)

```
typedef struct {
    char*         option;
    char*         specifier;
    XrmOptionKind argKind;
    XtPointer     value;
} XrmOptionDescRec, *XrmOptionDescList;
```

Fichier X Window nécessaire pour l'utilisation de cette structure

▷ `<X11/Xresource.h>`

Initialisation

Recherche de valeur de ressource sur la ligne de commandes

```
typedef enum {
    XrmoptionNoArg,
    XrmoptionIsArg,
    XrmoptionStickyArg,
    XrmoptionSepArg,
    XrmoptionResArg,
    XrmoptionSkipArg,
    XrmoptionSkipLine,
    XrmoptionSkipNArgs
} XrmOptionKind;
```

Initialisation

Exemple de définition d'options dans un programme d'application

```
...
XrmOptionDescRec options[] = {
 {"-verbose", "*verbose", XrmoptionNoArg, "TRUE"},
 {"-delay",   "*delay", XrmoptionSepArg, NULL }
};
...
int
main( int argc , char* argv[] ) {
  ...
  toplevel=XtInitialize( argv[0], "Debut", options, XtNumber(options), &argc, argv );
  ...
}
```

Initialisation de ressources sur l'application

```
{logname@hostname} debut facile -verbose
```

Gestion de Ressources

```
void XtSetArg( Arg warg, String resource_name, XtArgVal value )
```

Rôle

▷ Fixer des valeurs de ressources

Arguments

▷ `warg` : de type Intrinsics `Arg`

▷ `resource_name` : nom de ressource : Intrinsics (`XtN...`), Motif (`XmN...`)

▷ `value` : de type Intrinsics `XtArgVal`

Utilisation

```
XtSetArg(wargs[n], XmNlabelString, msg); n++;
```

Gestion de Ressources

Fichier X Window nécessaire : `<X11/Intrinsic.h>`

```
...
typedef long        XtArgVal;
...
typedef struct {
    String   name;
    XtArgVal value;
} Arg, *ArgList;
...
```

Définition de ressources Motif : `<Xm/XmStrDefs.h>`

```
...
#define XmNlabelString "labelString"
...
```

Création de widgets

```
Widget XtCreateManagedWidget( String name ,
                              WidgetClass class,
                              Widget parent,
                              Arglist wargs, Cardinal nwargs )
```

Rôle

▷ création, gestion de widget

▷ retourne une instance de widget

Appel des fonctions Intrinsics

▷ `XtCreateWidget()` : création

▷ `XtManageChild()` : visualisation

Création de widgets

Arguments

- ▷ `name` : nom du widget (accès ressources)
- ▷ `class` : pointeur sur la classe du widget
- ▷ `parent` : ascendant dans l'arbre des widgets
- ▷ `wargs , nwargs` : gestions de ressources sur le widget (`XtSetArg()`)

Utilisation

```
msg_widget= XtCreateManagedWidget( "msg", xmLabelWidgetClass, toplevel, wargs, n );
```

Fichier X Window nécessaire pour `xmLabelWidgetClass` : `<Xm/Label.h>`

Réalisation de widgets

```
void XtRealizeWidget( Widget w )
```

Rôle

▷ Création de fenêtre X associée aux widgets

Lorsqu'un widget est réalisé, tous les descendants qu'il gère le sont

Arguments

▷ `w` : le widget (l'arborescence) à réaliser

Utilisation

```
XtRealizeWidget(toplevel);
```

Boucle d'événements

`void XtMainLoop(void)`

Rôle

▷ gestion de la file d' événements du serveur X

Appel des fonctions Intrinsics

▷ `void XtNextEvent(XEvent* event)` : chercher le prochain événement

▷ `void XtDispatchEvent(XEvent* event)` : distribuer l'événement

Utilisation : `XtMainLoop();`

Boucle d'événements

Cette fonction revient à éxécuter la boucle

```
XEvent event;
while ( TRUE ) {
  XtNextEvent ( &event );
  XtDispatchEvent ( &event );
}
```

gérant eux-même les événements via la structure `XEvent` de la `Xlib`.
Cette boucle infinie devra parfois être gérée directement par le programmeur

Interactivité

Gestion de l'interactivité sous X Window

- ▷ évènementielle : **Events**
- ▷ réflexe sur widget : **Callbacks**
- ▷ correspondance sur widget : **Translations**

Fonction `Xt` de gestion d'interactivité

- ▷ `XtAddEventHandler()`
- ▷ `XtAddCallback()`
- ▷ `XtAugmentTranslations()`

Interactivité

Prototypes

```
Widget XtAddEventHandler( Widget w,
                          EventMask event_mask,
                          Boolean non_maskable,
                          XtEventHandler handler,
                          XtPointer client_data )
Widget XtAddCallback( Widget w,
                      String cb_resource ,
                      XtCallbackProc callback,
                      XtPointer client_data )

void XtAugmentTranslations( Widget w,
                            XtTranslations table)
```

Gestion d'événements

Fonction, action, à déclencher lors de l'interaction utilisateur

```
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include "libXs.h"

void quitEvent( Widget w, XtPointer client_data , XEvent* event,
                                                  Boolean* ctnd  )
{
  XtCloseDisplay( XtDisplay(w) );
  exit(0);
}
```

Gestion d'évènements

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

  Widget    toplevel , msg_widget;
  Arg       wargs[4];
  int       n;
  XmString  msg;

  toplevel=XtInitialize(argv[0],"Evenement",NULL,0,&argc,argv);
  n=0;
  XtSetArg(wargs[n], XmNwidth, 200); n++;
  XtSetArg(wargs[n], XmNheight, 50); n++;
  if ( (msg = XsConcatWords(argc-1,&argv[1]))!=NULL ) {
    XtSetArg(wargs[n], XmNlabelString, msg); n++;
  }
```

Gestion d'évènements

```
  msg_widget=XtCreateManagedWidget( "msg",
                                    xmLabelWidgetClass, toplevel,
                                    wargs, n);
  XtAddEventHandler( msg_widget,
                     ButtonPressMask, FALSE,
                     quitEvent, NULL);
  XtRealizeWidget(toplevel);
  XtMainLoop();
  return 0;
}
```

Sortie d'application lors d'un click de souris

Gestion d'évènements

Fonction de concaténation de chaîne Motif (libXs.h : fonctions utiles)

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>

XmString XsConcatWords(int n, char* words[]) {
  XmString xmstr;
  int i,len=0;

  if (n<=0) return XmStringCreate("",XmSTRING_DEFAULT_CHARSET);
  xmstr=(XmString) NULL;

  for(i=0;i<n;i++) {
    XmString tmp;
    if (i>0) {
      tmp=XmStringCreate("",XmSTRING_DEFAULT_CHARSET);
      xmstr=XmStringConcat(xmstr,tmp);
    }
    tmp=XmStringCreate(words[i],XmSTRING_DEFAULT_CHARSET);
    xmstr=XmStringConcat(xmstr,tmp);
  }
  return xmstr;
}
```

Gestion d'évènements

Premier programme : pas d'interactivité pour sortir de `XtMainLoop()`
Sortie d'application : gestion d'un événement (`ButtonPressMask`)

Lorsque l'utilisateur clique sur un bouton de la souris

- ▷ Appel de la fonction : `quitEvent()`
- ▷ Sortie du programme : `exit(0)`


```
{logname@hostname} evenement un peu plus complique
```

Gestion d'évènements

```
Widget XtAddEventHandler( Widget w,
                          EventMask event_mask,
                          Boolean non_maskable,
                          XtEventHandler handler,
                          XtPointer client_data )
```

Rôle

- ▷ gestion d' événement `event_mask`
- ▷ Déclenchement d' une action suite à une interaction `handler()`

Gestion d'évènements

Arguments

- ▷ `w` : widget sur lequel on gère l'évènement
- ▷ `event_mask` : masque d'évènement (`<X11/X.h>`)
- ▷ `non_maskable` : événement non-masquable
- ▷ `handler` : fonction à appeler
- ▷ `client_data` : données-client à transmettre

Utilisation

```
XtAddEventHandler( msg_widget, ButtonPressMask, FALSE, quitEvent, NULL);
```

Gestion d'évènements

Prototype d'une fonction d'évènement

```
typedef void (*XtEventHandler)( Widget w,
                                XtPointer client_data,
                                XEvent* event,
                                Boolean* continue_to_dispatch );
```

Utilisation d'une fonction d'évènement

```
void quitEvent( Widget w, XtPointer client_data, XEvent* event, Boolean* ctnd )
{
  XtCloseDisplay(XtDisplay(w));
  exit(0);
}
```

Gestion de Réflexes

Fonction, action, à déclencher lors de l'interaction utilisateur sur un widget

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include "libXs.h"

void quitCB( Widget w , XtPointer client_data , XtPointer call_data )
{
  XtCloseDisplay(XtDisplay(w));
  exit(0);
}
```

Gestion de Réflexes

```
int main(int argc,char* argv[])
{
  Widget    toplevel,msg_widget;
  Arg        wargs[4];
  int        n;
  XmString   msg;

  toplevel=XtInitialize(argv[0],"Reflexe",NULL,0,&argc,argv);
  n=0;
  XtSetArg(wargs[n], XmNwidth, 200); n++;
  XtSetArg(wargs[n], XmNheight, 50); n++;
  if ( (msg = XsConcatWords(argc-1,&argv[1]))!=NULL ) {
    XtSetArg(wargs[n], XmNlabelString, msg); n++;
  }
```

Gestion de Réflexes

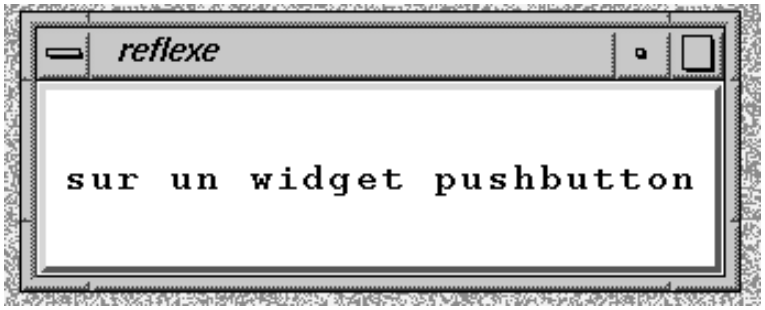
```
  msg_widget=XtCreateManagedWidget("msg",
                                   xmPushButtonWidgetClass,
                                   toplevel,
                                   wargs,
                                   n);
  XtAddCallback( msg_widget,
                 XmNactivateCallback,
                 quitCB, NULL);
  XtRealizeWidget(toplevel);
  XtMainLoop();
  return 0;
}
```

Gestion de Réflexes

Deuxième programme : sortie d'application par gestion d' événement
Troisième programme : gestion d'un réflexe prédéfini sur un widget

- ▷ Widget utilisé : `PushButton`
- ▷ réflexe associé : `XmNactivateCallback`

Gestion de Réflexes associée à un widget : `XmN...Callback`




```
{logname@hostname} reflexe sur un widget pushbutton
```

Gestion de Réflexes

```
Widget XtAddCallback( Widget w,
                      String cb_resource ,
                      XtCallbackProc callback,
                      XtPointer client_data )
```

Rôle

▷ gèrer un réflexe sur un widget

▷ ajouter une fonction réflexe à ce widget

Gestion de Réflexes

Arguments

- ▷ `w` : widget sur lequel on gère le réflexe
- ▷ `cb_resource` : ressource callback à gérer
- ▷ `callback` : Fonction à appeler
- ▷ `client_data` : Données client de l'application

Utilisation

```
XtAddCallback( msg_widget, XmNactivateCallback, quitCB , NULL );
```

Gestion de Réflexes

Prototype d'une fonction réflexe (fichier `<X11/Intrinsic.h>`)

```
typedef void (*XtCallbackProc) ( Widget w,
                                 XtPointer client_data,
                                 XtPointer call_data );
```

Utilisation d'une fonction réflexe

```
void quitCB( Widget w, XtPointer client_data, XmAnyCallbackStruct*  call_data )
{
  XtCloseDisplay(XtDisplay(w));
  exit(0);
}
```

Gestion de Réflexes

Structure de base lors de l'appel (fichier `<Xm/Xm.h>` pour un widget Motif)

```
typedef struct {
   int     reason;
   XEvent  *event;
} XmAnyCallbackStruct;
```

Chaque widget peut disposer d'une structure d'appel spécifique

- ▷ `XmPushButtonCallbackStruct`
- ▷ `XmToggleButtonCallbackStruct`
- ▷ `XmRowColumnCallbackStruct`
- ▷ `XmFileSelectionBoxCallbackStruct`

Correspondances

Fonction, action, à déclencher lors de l'interaction utilisateur sur un widget

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include "libXs.h"

void quitTrans( Widget w, XEvent* event,
                String* params, Cardinal* nparams )
{
  XtCloseDisplay(XtDisplay(w));
  exit(0);
}
```

Correspondances

```
static XtActionsRec actionsTable[]= { {"bye", quitTrans} };
static char defaultTranslations[]="<Key>Q:    bye()";

int main(int argc,char* argv[]) {
  Widget    toplevel,msg_widget;
  Arg        wargs[4];
  int        n;
  XmString   msg;
  XtTranslations   trans_table;

  toplevel=XtInitialize( argv[0], "Correspondance", NULL, 0,
                                                  &argc, argv );
  n=0;
  XtSetArg(wargs[n], XmNwidth, 200); n++;
  XtSetArg(wargs[n], XmNheight, 50); n++;
```

Correspondances

```
  if ( (msg = XsConcatWords(argc-1,&argv[1])) != NULL ) {
    XtSetArg( wargs[n] , XmNlabelString , msg ); n++;
  }
  msg_widget=XtCreateManagedWidget( "msg",
                                    xmLabelWidgetClass,
                                    toplevel, wargs, n );
  XtAddActions(actionsTable, XtNumber(actionsTable) );
  trans_table=XtParseTranslationTable (defaultTranslations );
  XtAugmentTranslations( msg_widget, trans_table );

  XtRealizeWidget(toplevel);
  XtMainLoop();
  return 0;
}
```

Correspondances

Gestion des correspondances

- ▷ rajouter une table d'actions sur une application
- ▷ associer des correspondances : `XtTranslations trans_table;`
- ▷ sur un widget : `XtAugmentTranslations(msg_widget,trans_table);`


```
{logname@hostname} correspondance par la touche Q
```

Intérêt des correspondances: interactions paramêtrables par un fichier ressource

Correspondances

Démarche à suivre pour établir une correspondance

1. définir une table d'actions

```
XtActionsRec actionsTable[]= { {"bye", quitTrans},};
```

2. définir les correspondances Interactivité / Action

```
char defaultTranslations[]="<key>Q:    bye()";
```

3. rajouter la table d'actions à l'application

```
XtAddActions(actionsTable, XtNumber(actionsTable) );
```

4. traduire les correspondances en table

```
XtTranslations tr_table=XtParseTranslationTable(defaultTranslations);
```

5. rajouter cette table de correspondances au widget concerné

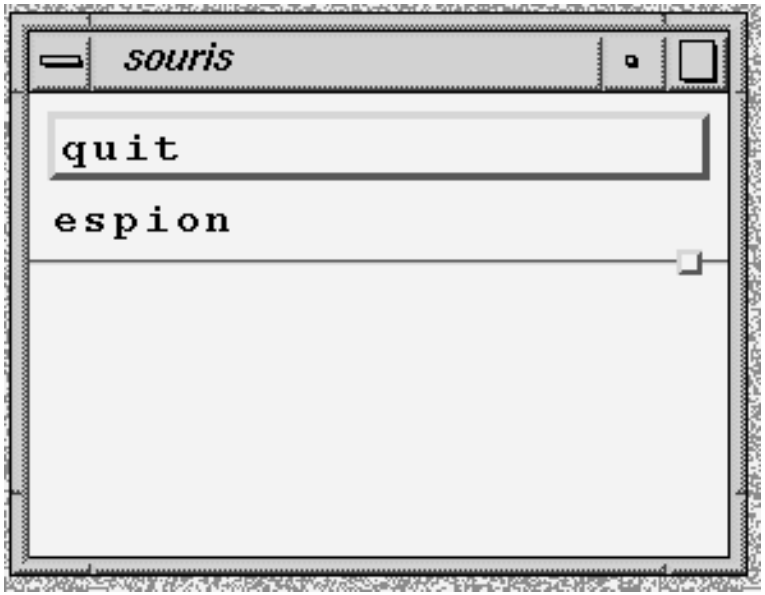
```
XtAugmentTranslations(msg_widget, trans_table);
```

Applications

Description de trois applications X Window Intrinsics / Motif

1. affichage des coordonnés de la souris
   - ▷ gestion de réflexes : sortie d'application (`XsQuit()`)
   - ▷ gestion d'évènements : affichage coordonnées (`creerEspion()`)
2. déplacement d'un widget dans une application
   - ▷ gestion de correspondances
3. chronomètre
   - ▷ gestion de tâches de fond

Affichage de Coordonnées



Affichage de Coordonnées

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/PanedW.h>
#include <Xm/RowColumn.h>
#include <Xm/DrawingA.h>
#include <Xm/Label.h>
#include "libXs.h"
#include "espion.h"

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

   Widget toplevel, panel, command, target;

   toplevel= XtInitialize( argv[0], "Souris", NULL, 0, &argc, argv );
```

Affichage de Coordonnées

```
    panel=   XtCreateManagedWidget( "panel",
                                    xmPanedWindowWidgetClass,
                                    toplevel, NULL, 0 );
    command= XtCreateManagedWidget( "command",
                                    xmRowColumnWidgetClass,
                                    panel, NULL , 0);
    XsQuit( command );
    target = XtCreateManagedWidget( "target",
                                    xmDrawingAreaWidgetClass,
                                    panel, NULL, 0);
    creerEspion( command, target ) ;
    XtRealizeWidget( toplevel );
    XtMainLoop();
    return 0;
}
```

Affichage de Coordonnées

Gestion de réflexes pour la sortie d'application (`libXs.h`)

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>

void armCB ( Widget w, XtPointer client_data, XtPointer call_data ){
 int* flag = (int *) client_data;
 *flag = FALSE;
}
void activateCB ( Widget w, XtPointer client_data, XtPointer call_data ){
  int* flag = (int *) client_data;
  *flag = TRUE;
}
```

Affichage de Coordonnées

```
void disarmCB ( Widget w, XtPointer client_data, XtPointer call_data ){
  int* flag = (int *) client_data;
  if(*flag){
    XtCloseDisplay(XtDisplay(w));
    exit(0);
  }
}
Widget XsQuit( Widget parent ) {
   Widget     w;
   static int really_quit;
   w=XtCreateManagedWidget("quit",xmPushButtonWidgetClass,parent,NULL,0);
   XtAddCallback( w, XmNarmCallback, armCB, &really_quit );
   XtAddCallback( w, XmNdisarmCallback, disarmCB, &really_quit );
   XtAddCallback( w, XmNactivateCallback, activateCB, &really_quit);
   return (w);
}
```

Affichage de Coordonnées

Gestion de sortie d'application:

- ▷ `ButtonPress` à l'intérieur du widget:
  - ◇ `XmNarmCallback`,initialisation de la donnée-client (`*flag=FALSE`)
- ▷ `ButtonRelease` à l'**intérieur** du widget:
  - ◇ `XmNactivateCallback`,modification de la donnée-client (`*flag=TRUE`)
- ▷ `ButtonRelease` à l'intérieur ou extérieur du widget:
  - ◇ `XmNdisarmCallback`: vérification de la donnée-client pour sortir

Intérêt de l' exemple:

- ▷ utilisation des données-clients associées aux fonctions réflexes

```
void callback( Widget w,
               XtPointer client_data,
               XtPointer call_data );
```

Affichage de Coordonnées

Gestion d'évènements pour afficher les coordonnées souris

- ▷ utilisation de l'union `XEvent` pour accéder aux coordonnées souris
- ▷ déplacement (`PointerMotionMask`) sur le widget où survient l'événement
- ▷ entrée/sortie (`LeaveWindowMask`) du curseur sur ce widget
- ▷ affichage dans un widget (`XmLabel`) transmis en données-client

```
void positionEvent(Widget w, XtPointer client_data, XEvent* event,
                                                    Boolean* cntd ) {
  Widget espion = (Widget) client_data;
  XsWprintf( espion, "X: %04d, Y: %04d",
             event->xmotion.x , event->xmotion.y);
}
```

Affichage de Coordonnées

```
void clearEvent(Widget w, XtPointer client_data, XEvent* event,
                                                  Boolean* cntd ) {
  Widget espion = (Widget) client_data;
  XsWprintf(espion, "espion" );
}

Widget creerEspion( Widget parent, Widget cible) {
  Widget espion;

  espion = XtCreateManagedWidget( "espion",
                                  xmLabelWidgetClass, parent, NULL, 0 );
  XtAddEventHandler( cible,
                     LeaveWindowMask, FALSE, clearEvent, espion );
  XtAddEventHandler( cible,
                     PointerMotionMask, FALSE, positionEvent, espion );
}
```

Affichage de Coordonnées

A titre indicatif on donne le code source (`libXs.h`)

```
#include <varargs.h>
#include <stdio.h>
...
void XsWprintf( va_alist ) va_dcl
{
  Widget    w;
  char*     format];
  va_list   args;
  char      str[1000;
  Arg       wargs[1];
  XmString  xmstr;

  va_start(args);
  w = va_arg(args, Widget);
  if(!XtIsSubclass(w, xmLabelWidgetClass)) XtError("XsWprintf() requires a Label Widget");
  format = va_arg(args, char *);
  vsprintf(str, format, args);
  xmstr =  XmStringLtoRCreate(str, XmSTRING_DEFAULT_CHARSET);
  XtSetArg(wargs[0], XmNlabelString, xmstr);
  XtSetValues(w, wargs, 1);
  va_end(args);
}
```

Affichage de Coordonnées

Dans ce programme on gère les évènements

- ▷ `LeaveWindowMask`
- ▷ `PointerMotionMask`

avec utilisation de la structure `XEvent`

```
void positionEvent( Widget w,
                    XtPointer client_data,
                    XEvent* event, Boolean* cntd )
{
  Widget espion = (Widget) client_data;
  XsWprintf( espion,
             "X: %04d, Y: %04d",
             event->xmotion.x , event->xmotion.y);
}
```

Affichage de Coordonnées

Le type `XEvent` est une union,
chaque champ correspond à une structure suivant l'évènement concerné

- ▷ `XButtonEvent xbutton;`
- ▷ `XMotionEvent xmotion;`
- ▷ `XCrossingEvent xcrossing;`
- ▷ ...

```
typedef union _XEvent {
  int type; /* must not be changed; first element */
  XAnyEvent xany;
  XKeyEvent xkey;
  XButtonEvent xbutton;
  XMotionEvent xmotion;
  XCrossingEvent xcrossing;
  ....
  ....
} XEvent;
```

Affichage de Coordonnées

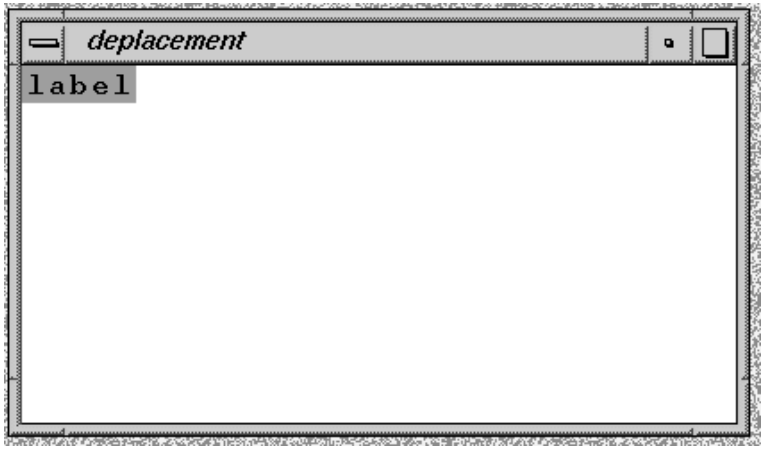
Structures définies dans l'en-tête : `<X11/Xlib.h>`

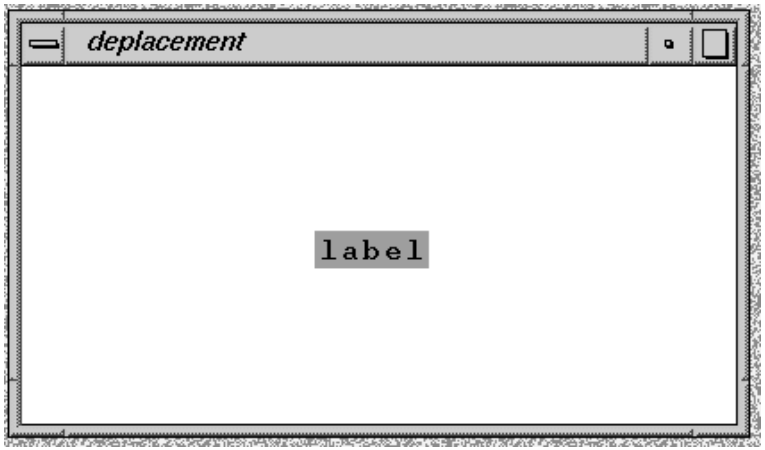
```
......

typedef struct {
  int type;                /* of event */
  unsigned long serial;    /* # of last request processed by server */
  Bool send_event;         /* true if this came from a SendEvent request */
  Display *display;        /* Display the event was read from */
  Window window;           /* "event" window it is reported relative to */
  Window root;             /* root window that the event occured on */
  Window subwindow;        /* child window */
  Time time;               /* milliseconds */
  int x, y;                /* pointer x, y coordinates in event window */
  int x_root, y_root;      /* coordinates relative to root */
  unsigned int state;      /* key or button mask */
  char is_hint;            /* detail */
  Bool same_screen;        /* same screen flag */
} XMotionEvent;
```

Affichage de Coordonnées

```
typedef struct {
  int type;                /* of event */
  unsigned long serial;    /* # of last request processed by server */
  Bool send_event;         /* true if this came from a SendEvent request */
  Display *display;        /* Display the event was read from */
  Window window;           /* "event" window reported relative to */
  Window root;             /* root window that the event occured on */
  Window subwindow;        /* child window */
  Time time;               /* milliseconds */
  int x, y;                /* pointer x, y coordinates in event window */
  int x_root, y_root;      /* coordinates relative to root */
  int mode;                /* NotifyNormal, NotifyGrab, NotifyUngrab */
  int detail;
    /*
     * NotifyAncestor, NotifyVirtual, NotifyInferior,
     * NotifyNonlinear,NotifyNonlinearVirtual
     */
  Bool same_screen;   /* same screen flag */
  Bool focus;         /* boolean focus */
  unsigned int state; /* key or button mask */
} XCrossingEvent;
typedef XCrossingEvent XEnterWindowEvent;
typedef XCrossingEvent XLeaveWindowEvent;
```

Déplacement de widget





Intérêt de ce programme

- ▷ gestion des correspondances
- ▷ système de fenêtrage X Window
- ▷ utilisation en-tête privée `IntrinsicP.h`

Déplacement de widget

```
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <X11/Composite.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>

Widget drag_label;
int drag_x_offset;
int drag_y_offset;

static void StartDragAction();
static void DragAction()
static void StopDragAction();
```

Déplacement de widget

```
static XtActionsRec  actions_table[]= {
  { "StartDrag" , StartDragAction },
  { "Drag"      , DragAction },
  { "StopDrag"  , StopDragAction },
};
static char label_trans[]=
  "<Btn2Down>: StartDrag() \n\
   <Motion>:   Drag() \n\
   <Btn2Up>:   StopDrag() \n\
  ";

int main( int argc , char* argv[]) {
  Widget toplevel, composite, label;

  toplevel= XtInitialize( argv[0], "Deplacement", NULL, 0, &argc, argv );
```

Déplacement de widget

```
  composite = XtVaCreateManagedWidget( "composite",
                                       compositeWidgetClass,  toplevel ,
                                       XtNwidth , 400,
                                       XtNheight, 200, NULL );
  label = XtVaCreateManagedWidget( "label",
                                   xmLabelWidgetClass, composite,
                                   XmNresize, 0, NULL );
  XtAddActions( actions_table, XtNumber(actions_table) );
  XtOverrideTranslations( label, XtParseTranslationTable(label_trans) );
  XtRealizeWidget(toplevel);
  XtMainLoop();
  return 0;
}
```

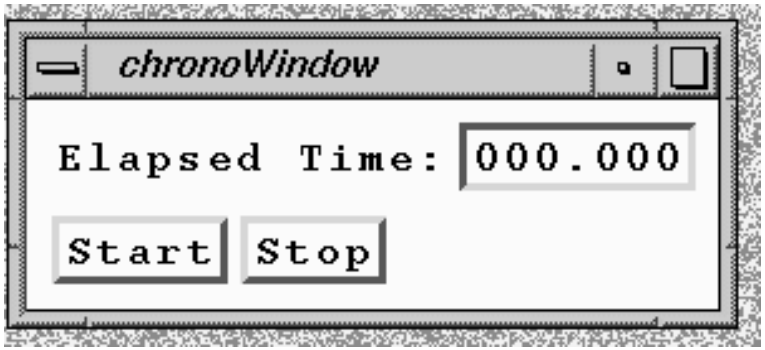
Déplacement de widget

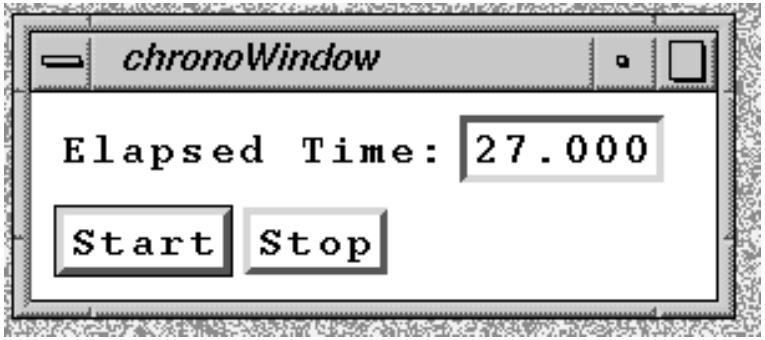
```
static void StartDragAction(Widget w , XEvent* event) {
  int drag_x;
  int drag_y;

  if (drag_label != NULL) return;
  drag_label = w;
  drag_x= w->core.x;
  drag_y= w->core.y;
  drag_x_offset= drag_x -event->xbutton.x_root;
  drag_y_offset= drag_y -event->xbutton.y_root;
}
```

Déplacement de widget

```
static void DragAction(Widget w , XEvent* event) {

  if (drag_label != NULL) {
     int x,y;
     x= event->xbutton.x_root + drag_x_offset;
     y= event->xbutton.y_root + drag_y_offset;
     if (x<0)  x=0;
     if (y<0)  y=0;
     XtMoveWidget (w,x,y);
   }
}
static void StopDragAction(Widget w , XEvent* event) {

     if (drag_label != NULL) drag_label = NULL;
}
```

Chronomètre





Procédure en tâche de fond: **Working Procedure**

▷ `Boolean functionWP ( XtPointer client_data)`

Enregistrement de la tâche de fond:

▷ `XtWorkProcId XtAddWorkProc( XtWorkProc workproc, XtPointer client_data)`

Annulation de procédure par son identificateur (`XtWorkProcId`)

▷ `XtRemoveWorkProc(XtWorkProcId id)`

Chronomètre

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>
#include <time.h>
#include "libXs.h"

Boolean updateTime();
void    startTiming();
void    stopTiming();

long         start_time;
XtWorkProcId work_proc_id = NULL;
```

Chronomètre

```
int main(int argc, char *argv[])
{
  Widget  toplevel, panneau, commandes;
  Widget  start, stop, horloge;

  toplevel  = XtInitialize (argv[0], "Chronometre", NULL, 0, &argc, argv);
  panneau   = XtCreateManagedWidget( "panneau",
                                      xmRowColumnWidgetClass,toplevel,
                                      NULL, 0 );
  horloge   = XtCreateManagedWidget( "horloge",
                                     xmLabelWidgetClass, panneau,
                                      NULL, 0 );
  commandes = XtCreateManagedWidget( "commandes",
                                     xmRowColumnWidgetClass, panneau,
                                      NULL, 0 );
```

Chronomètre

```
  start     = XtCreateManagedWidget( "start",
                                     xmPushButtonWidgetClass, commandes,
                                     NULL, 0 );
  XtAddCallback( start, XmNactivateCallback, startTiming, horloge );
  stop      = XtCreateManagedWidget( "stop",
                                     xmPushButtonWidgetClass, commandes,
                                      NULL, 0 );
  XtAddCallback( stop, XmNactivateCallback, stopTiming, horloge );

  XsQuit(commandes);
  XtRealizeWidget(toplevel);
  XtMainLoop();
  return 0;
}
```

Chronomètre

```
void startTiming( Widget w,
                  XtPointer client_data,XmAnyCallbackStruct *call_data){
  Widget horloge = ( Widget ) client_data;

  time( &start_time );
  if(work_proc_id) XtRemoveWorkProc( work_proc_id );
  work_proc_id = XtAddWorkProc( updateTime , horloge );
}

void stopTiming( Widget w,
                 XtPointer client_data,XmAnyCallbackStruct *call_data){
  Widget horloge = ( Widget ) client_data;

  if(work_proc_id) XtRemoveWorkProc(work_proc_id);
  work_proc_id = NULL;
}
```

Chronomètre

```
Boolean updateTime (Widget w )
{
  static long elapsed_time, last_time = -1;
  int minutes, secondes, current_time;

  time( &current_time );
  elapsed_time = current_time - start_time;
  if( last_time == elapsed_time ) return FALSE;

  last_time = elapsed_time;
  minutes = elapsed_time / 60;
  secondes = elapsed_time % 60;
  XsWprintf(w, "%02d : %02d", minutes, secondes);

  return FALSE;
}
```

Gestion de Ressources

Toute donnée paramêtrable par un utilisateur

- ▷ fenêtres, position, taille,
- ▷ texte, image,
- ▷ couleur, fonte,
- ▷ commande, etc...

Problème

- ▷ définir, accéder, modifier

les ressources, sur une application X Window, au niveau

- ▷ programmeur
- ▷ utilisateur

Gestion de Ressources

Dualité Programmeur / Utilisateur

- ▷ Quelle liberté doit-on laisser à l'**utilisateur** ?
- ▷ Peut-on adapter à nos besoins les ressources du **programmeur** ?

La Toolkit Intrinsics permet de

- ▷ Fixer des ressources dans une application
- ▷ Paramêtrer des ressources dans un fichier
- ▷ Définir des ressources sur la ligne de commandes
- ▷ Définir de nouvelles ressources

Accès dans un Programme

Un programmeur doit pouvoir **accéder, modifier** les ressources de widget

Trois fonctions Intrinsics

▷ `XtSetArg()`, `XtGetValues()`, `XtSetValues()`

permettent de

▷ fixer des **valeurs** de **ressource**

▷ accéder à des **ressources** de **widget**

▷ fixer des **ressources** de **widget**

Accès dans un Programme

```
void XtSetArg( Arg warg, String resource_name, XtArgVal value )
```

Rôle

▷ Fixer des valeurs de ressources

Arguments

▷ `warg` : de type Intrinsics `Arg`

▷ `resource_name` : nom de ressource

▷ `value` : de type Intrinsics `XtArgVal`

Utilisation

```
XtSetArg(wargs[n], XmNlabelString, msg); n++;
```

Accès dans un Programme

```
void XtGetValues( Widget w, ArgList wargs, Cardinal nwargs )
```

Rôle

▷ Connaître des ressources de widget

Arguments

▷ `w` : widget concerné

▷ `wargs` : tableau de ressources

▷ `nargs` : nombre de ressources du tableau

Utilisation

```
Dimension largeur;
n=0;
XtSetArg(wargs[n], XmNwidth, &largeur); n++;
XtGetValues(wbutton[1], wargs, n);
```

Accès dans un Programme

```
void XtSetValues( Widget w, ArgList wargs, Cardinal nwargs )
```

Rôle

▷ Fixer des ressources à un widget

Arguments

▷ `w` : widget concerné

▷ `wargs` : tableau de ressources

▷ `nwargs` : nombre de ressources du tableau

Utilisation

```
XtSetValues(wbutton[n], wargs, n);n++
```

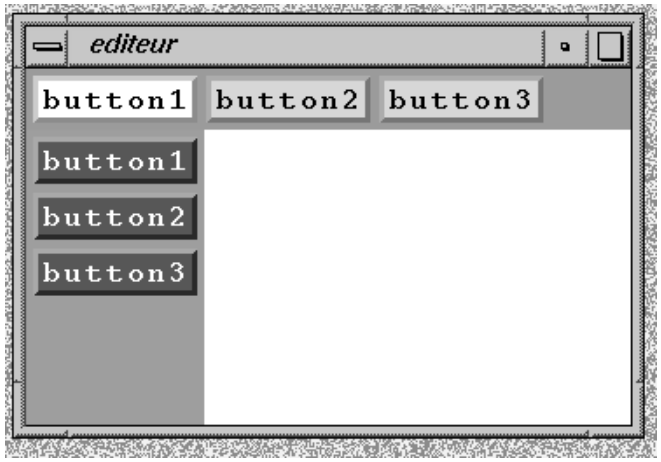
Accès dans un Programme

```
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
char* buttons[] = {"button1", "button2", "button3"};

main(int argc, char *argv[]) {
  Widget toplevel, form, wbutton[5];
  int    i, n=0;
  Arg wargs[10];
  toplevel = XtInitialize( argv[0], "Ressources", NULL, 0, &argc, argv );
  form = XtCreateManagedWidget( "form", xmFormWidgetClass, toplevel, NULL, 0 );
  for(i=0;i< XtNumber(buttons); i++)
    wbutton[i] = XtCreateWidget( buttons[i], xmPushButtonWidgetClass, form, NULL, 0 );
  XtManageChildren( wbutton, XtNumber(buttons) );
  XtSetArg(wargs[n], XmNtopAttachment,   XmATTACH_FORM);n++;
  XtSetArg(wargs[n], XmNleftAttachment,  XmATTACH_FORM);n++;
  XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM);n++;
  XtSetValues(wbutton[0], wargs, n);
```

Accès dans un Programme

```
  n=0;
  XtSetArg(wargs[n], XmNtopAttachment,   XmATTACH_WIDGET);n++;
  XtSetArg(wargs[n], XmNtopWidget,       wbutton[0]);     n++;
  XtSetArg(wargs[n], XmNleftAttachment,  XmATTACH_FORM);  n++;
  XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM);  n++;
  XtSetValues(wbutton[1], wargs, n);
  n=0;
  XtSetArg(wargs[n], XmNtopAttachment,   XmATTACH_WIDGET);n++;
  XtSetArg(wargs[n], XmNtopWidget,       wbutton[1]);     n++;
  XtSetArg(wargs[n], XmNleftAttachment,  XmATTACH_FORM);  n++;
  XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM);  n++;
  XtSetArg(wargs[n], XmNbottomAttachment,XmATTACH_FORM);  n++;
  XtSetValues(wbutton[2], wargs, n);

  XtRealizeWidget(toplevel);
  XtMainLoop();
}
```

Accès dans un Fichier

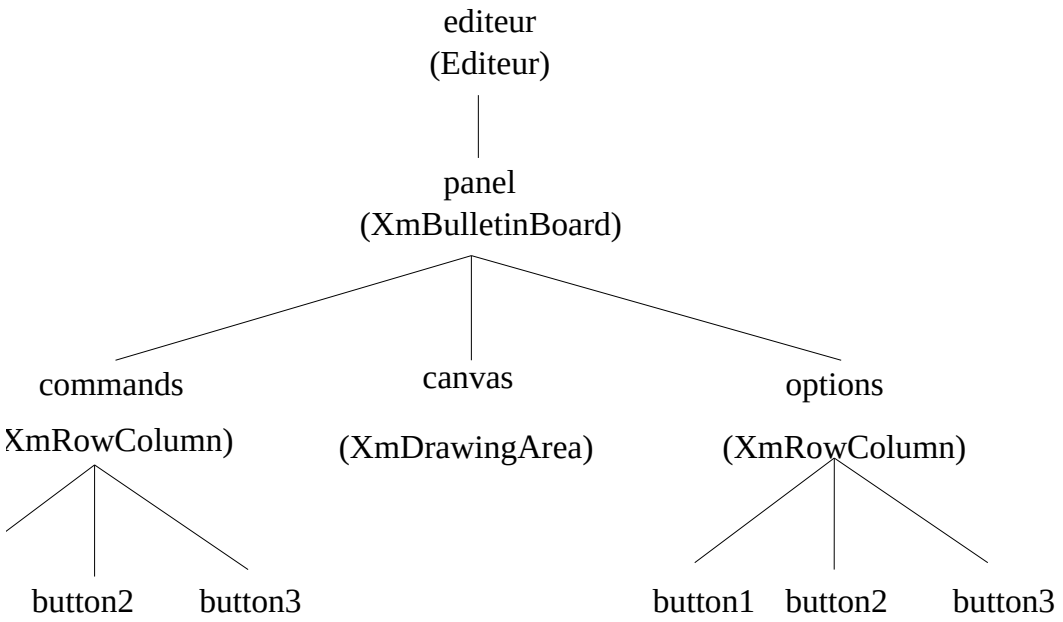



```
main(int argc, char *argv[]) {
...
 toplevel= XtInitialize(argv[0], "Editeur", NULL, 0, &argc, argv);
...
}
```

```
{logname@hostname} cc editeur.c -lXm -lXt -lX11 -o editeur
```

Accès dans un Fichier

Les ressources de widget peuvent-être paramêtrées par un utilisateur connaissant
l'**arborescence** des widgets de l'application



Accès dans un Fichier

```
!!!!!! Editeur : Fichier de ressources de l'application editeur !!!!!!!
Editeur*topAttachment:            attach_position
Editeur*bottomAttachment:         attach_position
Editeur*leftAttachment:           attach_position
Editeur*rightAttachment:          attach_position

Editeur*button1.leftPosition:      1
Editeur*button1.rightPosition:    99
Editeur*button1.topPosition:       1
Editeur*button1.bottomPosition:   31

Editeur*button2.leftPosition:     1
Editeur*button2.rightPosition:    99
Editeur*button2.topPosition:      35
Editeur*button2.bottomPosition:   65

Editeur*button3.leftPosition:      1
Editeur*button3.rightPosition:    99
Editeur*button3.topPosition:      69
Editeur*button3.bottomPosition:   99
```

Accès dans un Fichier

```
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
char* buttons[] = {"button1", "button2", "button3"};

main(int argc, char *argv[]) {
  Widget toplevel, form, wbutton[5];
  int    i, n;
  Arg wargs[10];
  toplevel = XtInitialize(argv[0], "Editeur", NULL, 0, &argc, argv);
  form = XtCreateManagedWidget("form", xmFormWidgetClass, toplevel, NULL,0);
  for(i=0;i< XtNumber(buttons); i++) {
    wbutton[i] = XtCreateWidget( buttons[i], xmPushButtonWidgetClass, form, NULL, 0 );
  }
  XtManageChildren(wbutton, XtNumber(buttons));
  XtRealizeWidget(toplevel);
  XtMainLoop();
}
```

Recherche

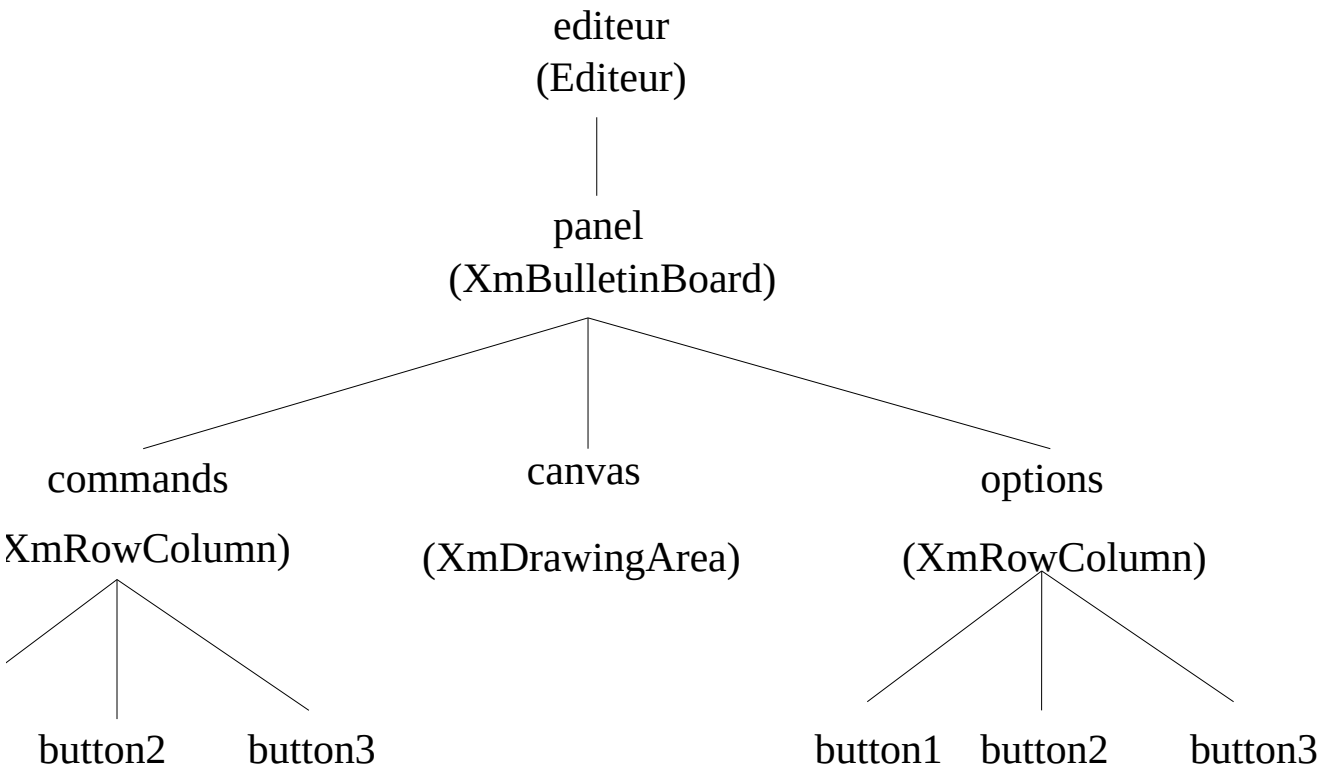
Le gestionnaire de ressources accède aux fichiers des répertoires

1. `/usr/lib/X11/<$LANG>/app-defaults/Filename`
2. `/usr/lib/X11/app-defaults/Filename`
3. `$XAPPLRESLANGPATH/Filename`
4. `$XAPPLRESDIR/Filename`
5. `$HOME/.Xdefaults`
6. `$HOME/Filename`

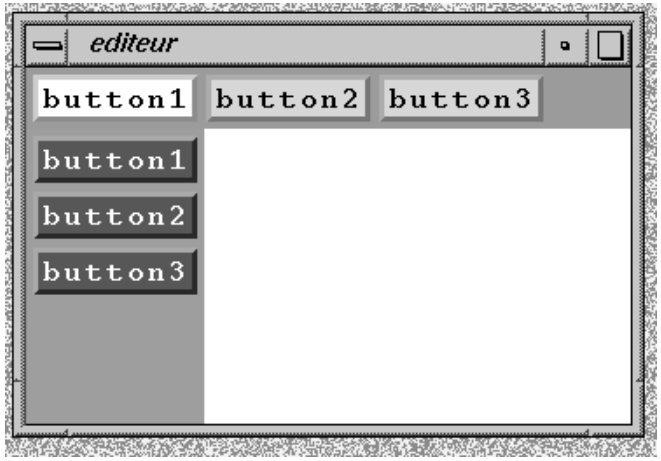
Priorité pour les ressources: la plus proche de l'utilisateur

Recherche

Accès aux ressources dans un même fichier



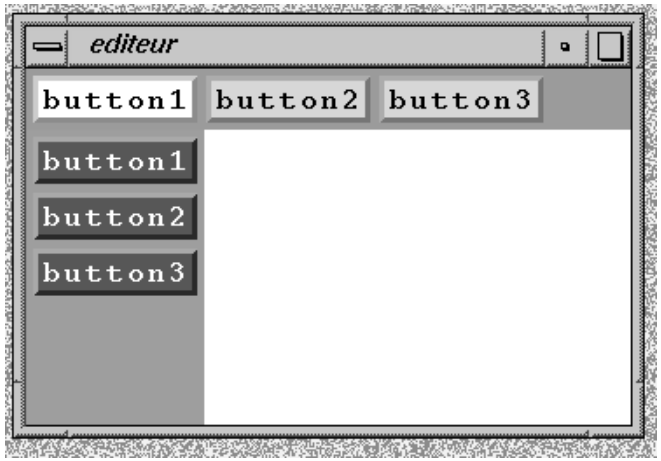
Recherche



Parcours d'arbre par instance prioritaire sur le parcours par classe

```
Editeur.panel.commands.button1.background : white
Editeur.panel.commands.button1.Background : green
Editeur.panel.commands.XmPushButton.Background : yellow
Editeur.XmBulletinBoard.XmRowColumn.XmPushButton.Background:red
```

Recherche



Parcours par chemin complet prioritaire

```
Editeur*button1.background : grey
Editeur*XmPushButton.background : red
Editeur*commands.button1.background : green
Editeur*XmRowColumn.XmPushButton.Background : yellow
```

Règle : ce qui est plus précis est prioritaire

Création de Ressources

La structure Intrinsics

```
typedef struct _XtResource {
    String      resource_name;
    String      resource_class;
    String      resource_type;
    Cardinal    resource_size;
    Cardinal    resource_offset;
    String      default_type;
    XtPointer   default_addr;
} XtResource, *XtResourceList;
```

permet de définir de nouvelles ressources dans une application

Création de Ressources

Une ressource est en fait un champ de structure, on doit

▷ se définir une structure de donnée

```
typedef struct {
    Pixel    fg, bg;
    int      delay;
    Boolean  verbose;
} ApplicationData, *ApplicationDataPtr;
```

▷ l'exploiter par le gestionnaire de ressources

```
static XtResource resources[] = {
{ XtNforeground, XtCForeground, XtRPixel, sizeof (Pixel),
  XtOffset(ApplicationDataPtr, fg), XtRString, "Black"     },
{ XtNbackground, XtCBackground, XtRPixel, sizeof (Pixel),
  XtOffset(ApplicationDataPtr, bg), XtRString, "White"     },
{ "delay", "Delay", XtRInt, sizeof (int),
  XtOffset(ApplicationDataPtr, delay),
  XtRImmediate, (caddr_t) 2},
{ "verbose", "Verbose", XtRBoolean, sizeof (Boolean),
  XtOffset(ApplicationDataPtr, verbose), XtRString, "FALSE"},
};
```

Création de Ressources

Charger les ressources dans l'application par la fonction `Xt`

```
void XtGetApplicationResources( Widget         w ,
                                XtPointer      base ,
                                XtResourceList resources ,
                                Cardinal       nresources ,
                                ArgList        args,
                                Cardinal       nargs )
```

Utilisation dans l'application

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

  ApplicationData data;
...
  XtGetApplicationResources( toplevel,
                             &data,
                             resources, XtNumber(resources),
                             NULL, 0);
...
}
```

Création de Ressources

```
void XtGetApplicationResources( Widget         w,
                                XtPointer      base,
                                XtResourceList resources, Cardinal nresources,
                                ArgList        args,      Cardinal nargs )
```

Rôle

Charger des ressources pour une application

Arguments

- ▷ `w` : Widget concerné
- ▷ `base` : du type de la structure de données
- ▷ `resources` : Tableau de ressources
- ▷ `nresources` : nombre de ressources
- ▷ `args, nargs` : cf `XtSetArg()`

Création de Ressources

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>

typedef struct {
   Pixel    fg, bg;
   int      delay;
   Boolean  verbose;
} ApplicationData, *ApplicationDataPtr;

static XtResource resources[] = {
{ XtNforeground, XtCForeground, XtRPixel, sizeof (Pixel),
  XtOffset(ApplicationDataPtr, fg), XtRString, "Black"     },
{ XtNbackground, XtCBackground, XtRPixel, sizeof (Pixel),
  XtOffset(ApplicationDataPtr, bg), XtRString, "White"     },
{ "delay", "Delay", XtRInt, sizeof (int),
  XtOffset(ApplicationDataPtr, delay),
  XtRImmediate, (caddr_t) 2},
{ "verbose", "Verbose", XtRBoolean, sizeof (Boolean),
  XtOffset(ApplicationDataPtr, verbose), XtRString, "FALSE"},
};
```

Création de Ressources

```
main(int argc,char* argv[])
{
  Widget    toplevel;
  ApplicationData data;

  toplevel=XtInitialize(argv[0], "Resappli", NULL, 0, &argc, argv);

  XtGetApplicationResources(toplevel, &data, resources, XtNumber(resources), NULL, 0);
  printf("fg = %d, bg = %d, delay = %d, verbose = %d\n", data.fg, data.bg, data.delay, data.verbose);
}
```

Exécution du programme

```
{logname@hostname} resappli
fg = 0 , bg = 1 , delay = 2 , verbose = 0
{logname@hostname}
```

Création de Ressources

Si on défini les ressources dans un fichier accessible `Resappli`

```
Resappli*foreground: red
Resappli*delay : 5
Resappli*verbose : TRUE
```

On obtient le résultat

```
{logname@hostname} resappli
fg = 9 , bg = 1 , delay = 5 , verbose = 1}
{logname@hostname}
```

Création de Ressources

```
Widget XtInitialize( String           name,
                     String           class_name,
                     XrmOptionDescRec options, Cardinal noptions,
                     Cardinal         *argc,   char **argv )
```

La structure du gestionnaire de ressources (Xrm..)

```
typedef struct {
    char            *option;
    char            *specifier;
    XrmOptionKind   argKind;
    XPointer        value;
} XrmOptionDescRec, *XrmOptionDescList;
```

permet de définir de nouvelles ressources (`<X11/resource.h>`)

Création de Ressources

Le type de recherche d'option `XrmOptionKind` peut-être

```
typedef enum {
    XrmoptionNoArg,
    XrmoptionIsArg,
    XrmoptionStickyArg,
    XrmoptionSepArg,
    XrmoptionResArg,
    XrmoptionSkipArg,
    XrmoptionSkipLine,
    XrmoptionSkipNArgs
} XrmOptionKind;
```

Exemple

- ▷ `XrmoptionNoArg` : option dans le champ `value` de `XrmOptionDescRec`
- ▷ `XrmoptionSepArg`: option dans l'argument suivant
- ▷ `XrmoptionIsArg` : la valeur de l'option est l'option elle-même

Création de Ressources

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>

typedef struct {
   Pixel    fg, bg;
   int      delay;
   Boolean  verbose;
} ApplicationData, *ApplicationDataPtr;
static XtResource resources[] = {
{ XtNforeground, XtCForeground, XtRPixel, sizeof (Pixel),
  XtOffset(ApplicationDataPtr, fg), XtRString, "Black"     },
{ XtNbackground, XtCBackground, XtRPixel, sizeof (Pixel),
  XtOffset(ApplicationDataPtr, bg), XtRString, "White"     },
{ "delay", "Delay", XtRInt, sizeof (int),
  XtOffset(ApplicationDataPtr, delay),
  XtRImmediate, (caddr_t) 2},
{ "verbose", "Verbose", XtRBoolean, sizeof (Boolean),
  XtOffset(ApplicationDataPtr, verbose), XtRString, "FALSE"},
};
```

Création de Ressources

```
XrmOptionDescRec options[] = {
  {"-verbose", "*verbose", XrmoptionNoArg, "TRUE"},
  {"-delay",   "*delay",   XrmoptionSepArg, NULL }
};

main(int argc,char* argv[])
{
  Widget    toplevel;
  ApplicationData data;

  toplevel=XtInitialize(argv[0], "Resoptions", options, noptions, &argc, argv);
  XtGetApplicationResources(toplevel, &data, resources, XtNumber(resources), NULL, 0);
  printf("fg = %d, bg = %d, delay = %d, verbose = %d\n", data.fg, data.bg, data.delay, data.verbose);
}
```

Création de Ressources

Si on défini les ressources dans un fichier `Resoptions`

```
Resoptions*foreground: red
Resoptions*delay : 5
Resoptions*verbose : FALSE
```

On obtient le résultat

```
{logname@hostname} resoptions
fg = 9 , bg = 1 , delay = 5 , verbose = 0
{logname@hostname}
```

Lancement de l'application avec options sur la ligne de commandes

```
{logname@hostname} resoptions  -delay 15  -verbose
fg = 9, bg = 1 , delay = 15 , verbose = 1
{logname@hostname}
```

Recherche de Fichiers

Les “include” **Intrinsics** se trouvent sous le répertoire

`/usr/include/X11`

L’en-tête `Intrinsic.h` contient les

- ▷ types
- ▷ structures
- ▷ déclarations de fonctions

nécessaires à toute application Intrinsics

Recherche de Fichiers

Les fichiers **Motif** se trouvent sous le répertoire

`/usr/include/Xm`

L'en-tête `Xm.h` est l'équivalent Motif de `Intrinsic.h`

Utilisation des commandes Unix de recherche

- ▷ Recherche de fichiers:

```
{logname@hostname} find /usr -name Xm.h -print
/usr/include/Xm/Xm.h
...
```

- ▷ Recherche de chaînes

```
{logname@hostname} cd /usr/include/Xm
{logname@hostname} grep XmCreateWidget *.h
ArrowB.h:extern Widget XmCreateArrowButton() ;
...
```

Recherche de Fichiers

Contenu de : `Intrinsic.h`

```
...

typedef unsigned int Cardinal;
typedef unsigned short Dimension;
typedef short Position;

...

typedef struct _XtResource {
    String resource_name;
    String resource_class;
    String resource_type;
    Cardinal resource_size;
    Cardinal resource_offset;
    String default_type;
    XtPointer default_addr;
} XtResource, *XtResourceList;
```

Recherche de Fichiers

```
...

extern Widget XtCreateWidget(
#if NeedFunctionPrototypes
    _Xconst _XtString  /* name */,
    WidgetClass  /* widget_class */,
    Widget  /* parent */,
    ArgList  /* args */,
    Cardinal  /* num_args */
#endif
);
...
```

Recherche de Fichiers

Contenu de `Xm.h`

```
...

typedef unsigned char * XmString;

...

typedef struct
{
    int     reason;
    XEvent  *event;
    Window  window;
} XmDrawingAreaCallbackStruct;

...
```

Recherche de Fichiers

```
#ifdef _NO_PROTO
extern Widget XmCreateSimpleMenuBar() ;
...
#else
extern Widget XmCreateSimpleMenuBar(
                        Widget parent,
                        String name,
                        ArgList args,
                        Cardinal arg_count) ;
...
#endif /* _NO_PROTO */
```

Recherche de Fichiers

```
...

typedef struct _CompositeClassRec *CompositeWidgetClass;

...

extern void XtManageChildren(
#if NeedFunctionPrototypes
    WidgetList  /* children */,
    Cardinal  /* num_children */
#endif
);

...

externalref WidgetClass compositeWidgetClass;

...
```

Recherche de Fichiers

```
externalref WidgetClass xmLabelWidgetClass;

typedef struct _XmLabelClassRec     * XmLabelWidgetClass;
typedef struct _XmLabelRec      * XmLabelWidget;

...

#ifdef _NO_PROTO

extern Widget XmCreateLabel() ;

#else

extern Widget XmCreateLabel(
                        Widget parent,
                        char *name,
                        Arg *arglist,
                        Cardinal argCount) ;
...
```

Bibliographie

O'Reilly
"Xlib Programming Manual Volume 1" O'Reilly & Associates, Inc. 1991

O'Reilly
"X Toolkit Intrinsics Programming Manual Volume 4" O'Reilly & Associates, Inc. 1991

Douglas A.Young
"The X Window System, Programming and Applications with Xt, OSF/Motif Edition" Prentice Hall 1990

Open Software Foundation
"OSF/Motif Programmer's Guide, Release 1.1" Prentice Hall 1991

C'EST FINI