Présentation Xt et Motif

Rappels

X Protocol = {les messages X échangés entre l'application et le serveur X}

Xlib = {fonctions en langage C correspondant à des messages du X protocol}

Xt/Intrinsics = le toolkit = boîte à outils de base (MIT) = {fonctions standards pour utiliser une toolkit X} = mécanismes de base d'objets graphiques

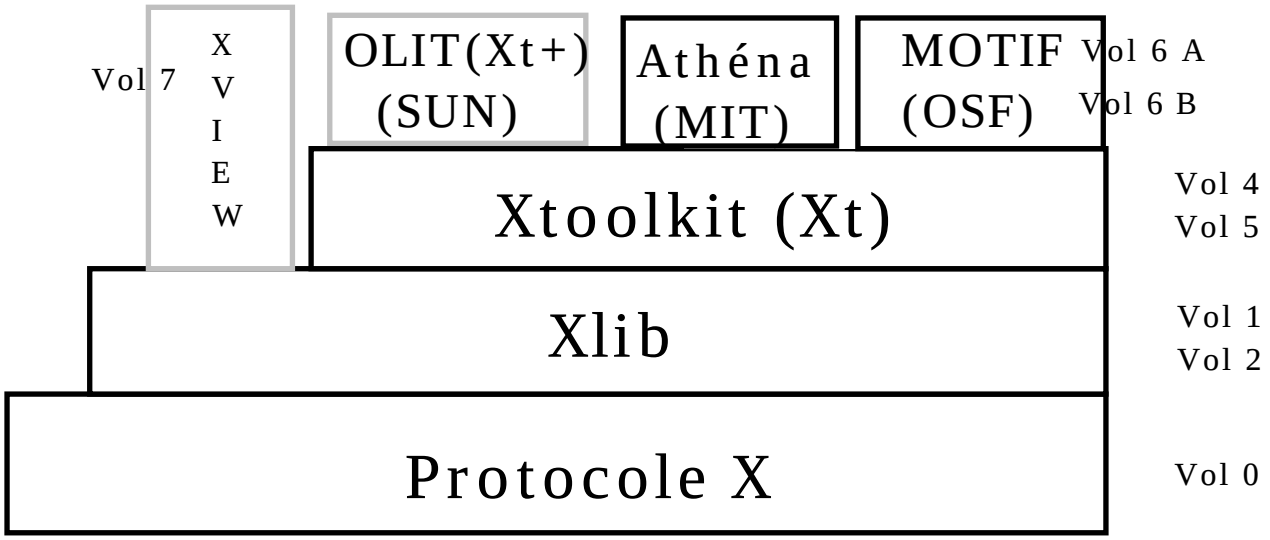
Toolkit Motif = la boîte à outils développée par OSF

autres toolkits : Athena (MIT), OLIT (= Xt+) (SUN, AT&T)

Constructeur d'interface : environ 200 sur le marché
- X-Designer (IST)
- IUM/X (Visual Edge)
- Architect
- XFM (NSL)
- environnement de développement de CDE.
- environnement multi-plateforme : Open Interface (Neuron Data), langage Java, ...

La "pile" X Window

Volume O'Reilly
and associates



Vol 3 : User's guide
Vol 8 : Administrator's guide

Structure d'un programme Xlib

```
main ( )
{
        Initialisation
        Création des fenêtres
        Boucle d'événements
}
```

L'essentiel du code est une grande boucle d'événements

Les fonctions de la Xlib commencent par X.
exemple : `XCreateSimpleWindow()`

Notion de widget

Widget = Window object
= un objet graphique

c'est-à-dire association de données (= ressources = paramètres de la widget) et de code.

Le code sera automatiquement lancé lors d'une certaine action de l'utilisateur.

De plus, Xt propose un mécanisme pour associer du code à certaines actions sur la widget : ce sont les listes de réflexes (ressources de callback). Évidemment cet autre code sera implanté par le programmeur.

En résumé, une partie du code est intrinsèque à la widget, l'autre partie est implantée par le programmeur et s'appelle une callback et Xt propose un mécanisme pour le lancer.

En français une callback est appelée un réflexe (D.A Young) ou une routine de rappel (N. Barkakati)

Structure d'un programme à l'aide d'une toolkit

```
fichiers de définition .h
main ( )
{
        Initialisation et création de la première widget
        Création des autres widgets avec leurs
        ressources
        Association de code pour ces widgets
        (callbacks)
        Gestion des widgets par leur mère
        Association widget-window
        boucle d'événements
}
Fonction button_pushed (....)
{
        /*   code lancé lors de l'appui du bouton
        (widget bouton poussoir)
        */
}
Fonction (....)
{
        /* code lancé et associé à une widget pour
        une certaine action de l'utilisateur (i.e. valeur de
        ressource de callback)
        */
}
```

Les fonctions et macros de la Xt commencent par Xt.

exemple : `XtManageChild()`, `XtNumber()`

Les fonctions Motif commence par Xm.

exemple : `XmCreateLabel()`

Structure d'un programme à l'aide d'une toolkit (suite)

"La toolkit Xt se charge elle-même de l'acheminement des événements"
Programmation style déclaratif : "lorsque A arrive, exécuter B" ou de style réactif : "lorsque l'action A se produit, lancer l'exécution de B"

Il y a alors une coupure nette entre la partie interface graphique de l'application (positionnement des widgets entre elles, dessin des fenêtres qui vont apparaître, ...) et le code des actions à lancer lorsqu'on agit sur telle ou telle widget.

En tête de programme Xt/Motif

Dès qu'un programme Motif a besoin d'une widget, il doit inclure le fichier .h contenant des définitions relatives à cette widget. Par exemple si on a besoin d'un bouton poussoir Motif il faut écrire :

```
#include <Xm/PushB.h>
```

On écrira donc des inclusions comme : `#include <Xm/`*`NomDeClasseWidget`*`.h>` pour chaque classe de widget utilisée dans le programme.

A partir de Motif 1.2, l'inclusion de ces fichiers suffisent.

Initialisation et création de la première widget

Création de la première widget de classe TopLevelShell par :

```
Widget XtVaAppInitialize (add_appctx,
classe_appli, TabOptions, NbOptions,
Adargc, argv, TabFallbackResources, ...,
NULL)
XtAppContext *add_appctx;
String classe_appli;
XrmOptionDescRec TabOptions[];
Cardinal NbOptions;
Cardinal *Adargc;
char *argv[];
String *TabFallbackResources;
```

XtVaAppInitialize() retourne l'identificateur de la widget créée (de type Widget)

add_appctx est un argument (ajouté dans X11R4) initialisé par cet appel. Ce contexte est initialisé par XtVaAppInitialize(&app, …) et utilisé par d'autres fonctions (comme XtAppMainLoop(app)).

classe_appli la classe de l'application.
Ce nom pourra être utilisé dans des fichiers de configurations (comme $HOME/.Xdefaults, ...).

Initialisation et création de la première widget (suite)

`classe_appli` permet de regrouper plusieurs applications ensemble (notion de classe) et de pouvoir ainsi les paramètrer de manière similaire. Par exemple `Xterm = {xterm, xpcterm, xaixterm}`

`TabOptions` est un tableau défini dans le programme indiquant que certains arguments de la ligne de commande seront traités d'une certaine manière pour configurer l'application. Les arguments traités (i.e. ceux qui étaient repérés par le tableau `TabOptions`) n'apparaissent plus dans `argv` après appel à `XtVaAppInitialize()` et `argc` est décrémenté d'autant.

`NbOptions` est le nombre d'éléments du tableau `TabOptions`.

`TabFallbackResources` est un tableau de chaînes de caractères terminé par `NULL` qui donnent des valeurs par défaut aux ressources des widgets de l'application.

Initialisation et création de la première widget (fin)

Remarque Fondamentale
Toutes les applications construites sur la Xt (donc les applications Motif), reconnaissent des arguments sur leur ligne de commande. Ce sont par exemple :

argument	Ressources
-background (-bg)	couleur de fond
-foreground (-fg)	couleur avant plan
-display	serveur X d'affichage
-geometry	l'emplacement
-iconic	lancement sous forme icône
-title	titre de l'application

exemple d'utilisation :

```
Widget toplevel ;
XtAppContext appctx;

      toplevel = XtVaAppInitialize (&appctx,
"MonAppli", NULL, 0, &argc, argv, NULL, NULL);
```

Remarques :
- La widget retournée par `XtVaAppInitialize()` ne peut contenir qu'une seule widget.
- `XtVaAppInitialize()` doit être appelée une et une seule fois dans l'application.

Création des autres widgets

obtenu à l'aide de la fonction :

```
Widget XtVaCreateWidget(String nomW,
WidgetClass classeWidget, Widget Wmere,
..., NULL)
```

Cette fonction retourne l'identificateur de la widget créée.

`nomW` est le nom de la widget créée (utile pour les fichiers de configuration).

`classeWidget` est la classe de la widget créée.

`Wmere` est la widget mère de la widget créée.

Suit une liste de paire ressource-valeur_de_cette_ressource.

exemple d'utilisation :

```
#include <Xm/PushB.h>

Widget Wcree, Wmere;

Wcree = XtVaCreateWidget ("mon_bouton",
        xmPushButtonWidgetClass, Wmere,
        XmNwidth, 300,
        XmNheight, 400,
        NULL);
```

Création des autres widgets (suite)

Pour la plupart des widgets, Motif propose des fonctions plus lisibles que `XtVaCreateWidget()`. Ces fonctions sont de la forme :

`XmCreate<`*`Nom de classe de widget`*`>`

Par exemple :

`XmCreatePushButton()`

Attention

Ces fonctions utilisent un tableau de paire ressource-valeur (cf. appendice). De plus l'ordre et le nombre des arguments n'est pas le même que dans `XtVaCreateWidget()`.

```
#include <Xm/PushB.h>
Widget XmCreatePushButton(Widget
    widgetmere;
    String nom_de_la_widget;
    ArgList TabListeArguments;
    Cardinal nbEltsTabListeArguments);
```

exemple d'utilisation :

```
#include <Xm/PushB.h>
int n;
Widget Wcree, Wmere;
Arg args[10];
n = 0;
XtSetArg(args[n], XmNwidth, 300); n++;
XtSetArg(args[n], XmNheight, 400); n++;
Wcree = XmCreatePushButton (Wmere,
      "mon_bouton", args, n);
```

La gestion des widgets

Toutes les widgets d'une application sauf les widgets shell sont contenues et donc gérées par d'autres widgets. C'est la widget contenante (mère) qui gère la redirection des E/S, la position, ... des widgets contenues (filles).

Il faut indiquer à une widget contenante qu'elle possède des widgets filles. On utilise la fonction :

```
void XtManageChild (Widget wfille);
```

exemple d'utilisation :

```
Widget Wmere, Wcree;
Wcree = XtCreateWidget(•••, Wmere, •••);
XtManageChild(Wcree);
```

Remarques importantes

1°) C'est cette fonction qui rend une widget visible à l'écran. Pour faire disparaître à l'écran une widget on utilise :

```
void XtUnmanageChild (Widget wfille);
```

2°) Manager une widget indique seulement son existence à sa mère directe (pas de récursité ici). Il faut donc faire des appels à `XtManageChild()` à chaque niveau de l'arborescence des widgets si on veut les rendre visibles.

La gestion des widgets (suite)

Lorsqu'une widget possède plusieurs widgets filles repérées par un tableau, on peut les "manager" avec un seul appel par :

```
void XtManageChildren(Widget TabWid[],
Cardinal NbEltTabWid);
```

De même on fait disparaître de l'écran l'ensemble de ces widgets par :

```
void XtUnmanageChildren(Widget TabWid[],
Cardinal NbEltTabWid);
```

(une seule requête au serveur X).

Création et gestion de widgets

On peut, par un seul appel, créer et gérer une widget par sa mère. On utilise :

```
Widget XtVaCreateManagedWidget(
    String nomW,
    WidgetClass classeWidget,
    Widget Wmere,
    ...,
    NULL)
```

remarque :
Il est conseillé de le faire pour les widgets de classes sous classes de la classe `XmPrimitive` mais par pour les autres widgets.

Association widget-window

Pour des raisons d'efficacité, l'association des widgets (notion Xt) à des fenêtres (notion Xlib) doit être faite explicitement par l'appel :

```
XtRealizeWidget(Widget wid);
```

qui affiche toute l'arborescence (récursivité) à partir de la widget `wid`. En général c'est la widget racine.

exemple d'utilisation :

```
Widget racine;
racine = XtVaAppInitialize(...);
...
XtRealizeWidget(racine);
```

La "boucle d'événements" Xt

Elle est lancée par :

```
XtAppMainLoop(XtAppContext appctx);
```

où `appctx` est l'argument initialisé par `XtVaAppInitialize()`

exemple d'utilisation :

```
XtContext appctx;
... = XtVaAppInitialize(&appctx, ...);
...
XtAppMainLoop(appctx);
```

exemple de programme Motif

```
#include <Xm/Label.h>

main(argc, argv)
int    argc;
char  *argv[];
{
Widget racine , mon_label;
XtAppContext appctx;

  racine = XtVaAppInitialize(&appctx, "Hello",
              NULL, 0,
              &argc, argv,
              NULL,
              NULL);

mon_label = XtVaCreateManagedWidget(
       "message",
       xmLabelWidgetClass, racine, NULL);

XtRealizeWidget(racine );
XtAppMainLoop(appctx);
}
```

compiler un programme motif

En général la commande de compilation est :
`cc fic_source.c -lXm -lXt -lX11 -o fic_exec`

Positionnement de valeur à une ressource

On peut donner des valeurs aux ressources autrement qu'à la création de la widget grâce à :

```
XtVaSetValues (Widget w,
    liste-paire-ressource-valeur,
    NULL);
```

exemple :

```
Widget w_mere;

w_pb = XtVaCreateWidget ("mon_bouton",
xmPushButtonWidgetClass, w_mere, NULL);

XtVaSetValues(w_pb ,
            XmNwidth, 300,
            XmNheight, 400,
            NULL);
```

Les ressources

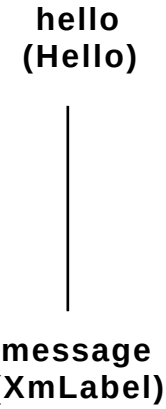
Les valeurs de ressources des widgets sont souvent données par des fichiers de configuration et non pas codées "en dur" dans le programme. Cela permet à chaque utilisateur de personnaliser son application pour les ressources non fondamentales (couleur de bouton, ...).

Pour paraphraser l'"équation" de N. Wirth (et puisque les ressources sont les données), on a :

exécutable + fichier de ressources = programme Xt/Motif

arborescence des widgets

Si le programme exécutable se nomme `hello` (= `argv[0]`), l'arborescence des widgets du programme ci dessus est :



On note le nom des widgets et en dessous (et entre parenthèses) le nom de la classe de la widget.
Ces noms sont "externes" au programme c'est à dire pourront être utilisés pour configurer le programme par un utilisateur (même ci celui-ci ne connaît pas le langage C !!)

Par exemple on peut mettre un libellé dans le label en écrivant dans un fichier de configuration :

```
hello.message.labelString: BONJOUR
```

remarque
Dans un programme un nom de ressource est de la forme `XmN<`*`ressource`*`>`. C'est en fait un

```
#define XmN<ressource> "<ressource>"
```

=> vérification par le compilateur
Dans un fichier de ressources seul le nom *`<ressource>`* est utilisé.

Fixer les ressources d'une widget

Quand on lance l'exécution d'une application, l'appel à `XtVaAppInitialize()` lit différentes sources d'information pour fixer des valeurs de ressources aux widgets, fait une fusion de toutes ces valeurs et fabrique avec cela une BD interne.
Les "différentes sources d'information" sont :
- des fichiers de ressources
- la propriété `RESOURCE_MANAGER`
- les options passées sur la ligne de commande.

A chaque création de widget pendant l'exécution du programme, cette BD est lue et permet de positionner des valeurs à la widget en cours de création. Les valeurs données par le programme lui-même sont prioritaires sur celles obtenues par la BD.

Fixer les ressources d'une widget (suite)

Les valeurs mises dans la BD sont obtenues dans l'ordre de priorité croissante ci dessous (i.e. si de nouvelles valeurs sont définies, elles écrasent les anciennes) :

1°) Dans le fichier /usr/lib/X11/$*LANG*app-defaults/**class_appli** où *LANG* est une variable d'environnement dont le nom doit se terminer par /.

2°) Dans le fichier $*XAPPLRESLANGPATH***class_appli**. Si *XAPPLRESLANGPATH* n'existe pas ou si le fichier est introuvable, Motif recherche le fichier $*XAPPLRESDIR***class_appli**

3°) Dans les données de la propriété RESOURCE_MANAGER (positionné par xrdb) associée à la fenêtre racine. Si cette propriété n'existe pas, charger les ressources définies dans $HOME/.Xdefaults

4°) Dans le fichier $XENVIRONMENT. S'il n'existe pas, charger le contenu du fichier $HOME/.Xdefaults-*Machine_hote*

5°) Les options de la ligne de commande (après l'option -xrm).

Rappelons que ces 5 étapes construisent la BD interne mais ces valeurs sont moins prioritaires que celles qui sont fixées par le programme.

Hiérarchie des widgets dans une application

Une widget (hormis la première de l'application) est toujours créée comme fille d'une autre widget => structure arborescence "géométrique" des widgets

Exemple :

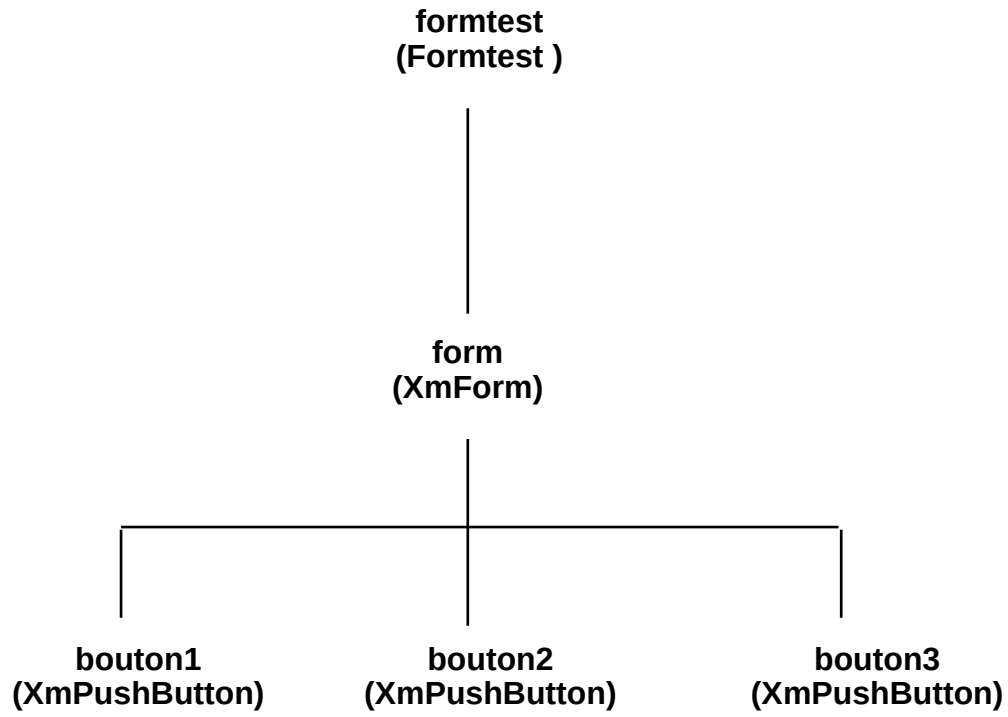
```
#include <Xm/Form.h>
#include <Xm/PushB.h>
char * boutons[ ] = {"bouton1", "bouton2", "bouton3"};

main(argc, argv)
int   argc;
char *argv[ ];
{
    XtAppContext ctx;
    Widget w_toplevel, w_form, w_bouton[3];
    int    i;

w_toplevel = XtVaAppInitialize(&ctx, "Formtest", NULL,
    0, &argc, argv, NULL, NULL);
w_form = XtVaCreateWidget("form",
    xmFormWidgetClass, w_toplevel, NULL);
for(i=0; i< XtNumber(boutons); i++)
    w_bouton[i] = XtVaCreateWidget(boutons[i],
          xmPushButtonWidgetClass,
          w_form, NULL);
XtManageChildren(w_bouton, XtNumber(w_bouton));
XtManageChild(w_form);
XtRealizeWidget(w_toplevel);
XtAppMainLoop(ctx);
}
```

Hiérarchie des widgets dans une application (suite)

La hiérarchie "géométrique" du programme boutons est :



Configuration à l'aide de fichier de ressources

L'arborescence des widgets est utilisée, entre autre, pour configurer l'application.

Un fichier de ressources est une BD contenant des lignes de la forme :

```
branche-menant-à-la-Widget . Nom-de-la-ressource : Valeur
```

Dans `branche-menant-à-la-widget`, on peut mettre le chemin complet de l'arborescence par exemple :

```
formtest.form.bouton3.labelString :     bonjour
```

ou utiliser des raccourcis en mettant une *

Exemple :

```
formtest*bouton3.labelString :     bonjour
```

On peut aussi mettre des noms de classe de widget.

Pour charger le fichier de ressources au niveau du serveur, on utilise le programme `xrdb`

Exemple :

fichier de ressources `formtest.rdb` chargé par `xrdb formtest.rdb`

=> configuration de l'application SANS recompilation : Personnalisation par l'utilisateur (Customization)

Exemple : fichier formtest.rdb

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! formtest.rdb : fichier de ressources pour application
! formtest
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
Formtest*topAttachment:          attach_position
Formtest*bottomAttachment:       attach_position
Formtest*leftAttachment:         attach_position
Formtest*rightAttachment:        attach_position
!
! Les boutons occuperont toute la widget form
! chaque bouton occupe 30% de la hauteur de la
! forme
!
Formtest*bouton1.leftPosition:    1
Formtest*bouton1.rightPosition:   99
Formtest*bouton1.topPosition:     1
Formtest*bouton1.bottomPosition:  31

Formtest*bouton2.leftPosition:    1
Formtest*bouton2.rightPosition:   99
Formtest*bouton2.topPosition:     35
Formtest*bouton2.bottomPosition:  65

Formtest*bouton3.leftPosition:    1
Formtest*bouton3.rightPosition:   99
Formtest*bouton3.topPosition:     69
Formtest*bouton3.bottomPosition:  99
```

Ces valeurs de ressources sont celles d'une widget de classe XmForm. Elles sont utilisées pour positionner les widgets qu'elle contient.

Appendice

Programmation sans les "Va et les app"

Structure d'un programme à l'aide d'une toolkit

```
fichiers de définition .h

main ( )
{
        Initialisation et création de la première widget
        Création des autres widgets avec leurs
        ressources
        Association de code a ces widgets (callbacks)
        Gestion des widgets par leur mère
        Affichage des widgets
        Boucle d'événements Xt
}

Fonction button_pushed (....)
{
        /*   code lancé lors de l'appui du bouton
        (widget PushButton)
        */
}
```

Initialisation et création de la première widget

Création de la première widget de classe TopLevelShell par :

```
Widget XtInitialize (nom_appli,
classe_appli, TabOptions, NbOptions,
Adargc, argv)
char *nom_appli;
char *classe_appli;
XrmOptionDescRec  TabOptions[];
Cardinal NbOptions;
Cardinal *Adargc;
char *argv[];
```

XtInitialize() doit être appelée une et une seule fois dans l'application.

nom_appli est le nom de l'application et classe_appli la classe de l'application.
Ces 2 noms pourront être utilisés dans des fichiers de configurations (comme $HOME/.Xdefaults, ...).

classe_appli permet de regrouper plusieurs applications ensemble (notion de classe) et de pouvoir ainsi les paramètrer de manière similaire. Par exemple Xterm = {xterm, xpcterm, xaixterm}

Initialisation et création de la première widget (suite)

`nom_appli` commence par une minuscule, `classe_appli` commence par une majuscule.
`TabOptions` est un tableau défini dans le programme indiquant que certains arguments de la ligne de commande seront traités d'une certaine manière pour configurer l'application. Les arguments traités (i.e. ceux qui étaient repérer par le tableau `TabOptions`) n'apparaissent plus dans `argv` après appel à cette fonction.
`NbOptions` est le nombre d'éléments du tableau `TabOptions`.

exemple d'utilisation :

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

La widget retournée par `XtInitialize()` de type `TopLevelShell` ne peut contenir qu'une seule widget.

Création des autres widgets

On crée les autres widgets par

```
Widget XtCreateWidget (nomW, classeW,
          Wmere, args, nargs)
String nomW;
WidgetClass classeW;
Widget Wmere;
ArgList args;
Cardinal nargs;
```

`nomW` est le nom de la widget. Ce nom sert pour configurer cette widget dans les fichiers de ressources.

`classeW` est la classe de la widget (i.e. son type)

`Wmere` est la widget mère (i.e. "contenante") de la widget qu'on est en train de définir.

`args` est le tableau des ressources qu'on veut positionner dans cette widget. Ces valeurs ont été mises dans ce tableau `args` par la macro `XtSetArg`

`nargs` est le nombre d'éléments du tableau `args`

Création des autres Widgets (suite)

exemple :

```
Arg args[10];
int n;
Widget mere;

...

n=0;
XtSetArg (args[n], XmNwidth, 300); n++;
XtSetArg (args[n], XmNheight, 400); n++;
XtCreateWidget ("mon_bouton",
xmPushButtonWidgetClass, mere, args, n);
```

On sait d'après la documentation Motif (O'Reilly Vol 6B) qu'une widget de la classe `XmPushButton` (i.e. un bouton) possède les ressources `XmNwidth`, `XmNheight`.

En résumé :
`XtSetArg` remplit chaque case du tableau `args`

Création de widgets avec Motif

Pour la plupart des widgets, Motif propose des fonctions plus lisibles que `XtCreateWidget()`. Ces fonctions sont de la forme :
`XmCreate<`*`Nom de classe de widget`*`>`
Par exemple :
`XmCreatePushButton()`
Attention à l'ordre et au nombre des arguments qui n'est pas le même que dans `XtCreateWidget()`. Par exemple :

```
#include <Xm/PushB.h>

Widget XmCreatePushButton(
    widgetmere, nom_de_la_widget,
    TabListeArguments,
    nbEltsTabListeArguments )
Widget widgetmere;
String nom_de_la_widget;
ArgList TabListeArguments;
Cardinal  nbEltsTabListeArguments;
```

Retour sur XtSetArg

Bien que présentée comme une fonction,

```
void XtSetArg(arg, NomDeRessource, valeur)
Arg arg;
String NomDeRessource;
XtArgVal valeur;
```

où Arg est le type

```
typedef struct {
    String name;
    XtArgVal value;
} Arg;
```

c'est en fait une macro .

```
#define XtSetArg(arg,n,d) \
((arg).name = (n), (arg).value = (XtArgVal)(d))
```

d'où problème si arg vaut args[n] et qu'on écrit args[n++] : l'itération sera effectuée 2 fois.

Remarque :

Cette macro permet de travailler dans les 2 sens !! Pour positionner des ressources cf. ci dessus. Pour récupérer des valeurs de ressources :

```
Arg args[10];
int n ;
Dimension  longW, hauteurW;
Widget w;

n = 0;
XtSetArg (args[n], XmNwidth, &longW); n++;
XtSetArg (args[n], XmNheight, &hauteurW); n++;
XtGetValues(w, args, n);
printf("widget w de largeur = %d\n", longW);
printf("widget w de largeur = %d\n", hauteurW);
```

XtSetValues

On peut positionner des valeurs de ressources à une widget déjà créée grâce à la macro `XtSetValues (Widget w, ArgList args, Cardinal Nbargs)`

exemple :

```
Arg args[10];
int n;
Widget mere;

pbw = XtCreateWidget ("mon_bouton",
xmPushButtonWidgetClass, mere, NULL, 0);

n=0;
XtSetArg (args[n], XmNwidth, 300); n++;
XtSetArg (args[n], XmNheight, 400); n++;

XtSetValues(pbw, args, n);
```

La gestion et création des widgets sans les Va

La gestion (information de son existence à sa mère) et la création d'une widget peut être faîte avec une fonction "sans Va" par :

```
XtCreateManagedWidget(NomW, NomClasseW,
W_mere, args, nargs);
```

La "boucle d'événements" Xt

La fonction sans App (donc sans contexte) est :

```
XtMainLoop();
```

exemple de programme Xt "sans Va ni App"

```
#include <Xm/Label.h>

main(argc, argv)
int    argc;
char  *argv[];
{
Widget w_racine , w_label;
Arg args[10];
int    n ;
XmString chaine;

w_racine = XtInitialize(argv[0], "Hello", NULL, 0,
&argc, argv);
/*
 * Crée une widget label pour un message.
 */
chaine = XmStringCreate("Premier Programme Xt",
XmSTRING_DEFAULT_CHARSET);

n=0;
XtSetArg(args[n], XmNlabelString, chaine); n++;
w_label = XtCreateManagedWidget("message",
        xmLabelWidgetClass,racine, args, n);

XtRealizeWidget(w_racine );
XtMainLoop();
}
```

compiler un programme motif

En général la commande de compilation est :

```
% cc fic_source.c -lXm -lXt -lX11 -o fic_exec
```