MASTER EN SCIENCES INFORMATIQUES

Spécification et conception d'objets interactifs pour l'environnement Windows 3.0

Bawin, Paul

Award date:
1992

Awarding institution:
Universite de Namur

Facultés Universitaires N.D. de la Paix à Namur

Institut d'Informatique

rue Grandgagnage, 21-B 5000 Namur

Spécification et conception d'objets interactifs pour l'environnement Windows 3.0

Paul BAWIN

Mémoire présenté en vue de l'obtention du titre de Licencié et Maître en Informatique

Promoteur : Monsieur le Professeur François BODART (Institut d'Informatique).

Remerciemment

Au terme de ce travail, nous tenons à exprimer notre gratitude envers les personnes qui nous ont aidé et conseillé tout au long de son élaboration.

Nos remerciements s'adressent particulièrement :

à Monsieur le Professeur F.BODART qui nous a encouragé et guidé lors de la réalisation du travail;

à Monsieur L.GILAIN pour son aide et ses conseils pertinents;

à Monsieur B.SACRE pour ses remarques judicieuses;

à Monsieur J.M.LEHEUREUX pour sa gentillesse, sa disponibilté;

à mon amie pour son soutien;

ainsi qu'à mes proches pour leur aide et leur tolérance.

Avant propos

Je suis entré dans la société *Multiscreen* (*MTS*) il y a plus de trois ans après avoir effectué un stage européen (COMETT) dans une société française. *MTS* est une société de service qui s'est spécialisée dans le développement de logiciels techniques et scientifiques. Très vite, nous nous sommes rendu compte de l'importance de se doter d'outils pour la conception des interfaces utilisateurs, et nous avons choisi MS *Windows* 2.0 comme support. Ce choix a été confirmé par l'arrivée de la version 3.0 de ce produit et par l'influence qu'il exerce aujourd'hui sur le monde micro et en particulier sur le monde PC. Cet environnement s'est avéré puissant mais d'un temps de maîtrise relativement long.

Pour ma part, voilà un an et demi que je développe dans cet environnement. Auparavant, j'ai eu l'occasion de participer à un travail de recherche sur la programmation par objets.

Me rendant compte qu'il me manquait un certain formalisme dans le domaine de la science de l'information, j'ai entrepris une licence & maîtrise en informatique aux Facultés Universitaires Notre-Dame de la Paix à Namur. Dans le cadre de ces études j'ai dû choisir un sujet de mémoire.

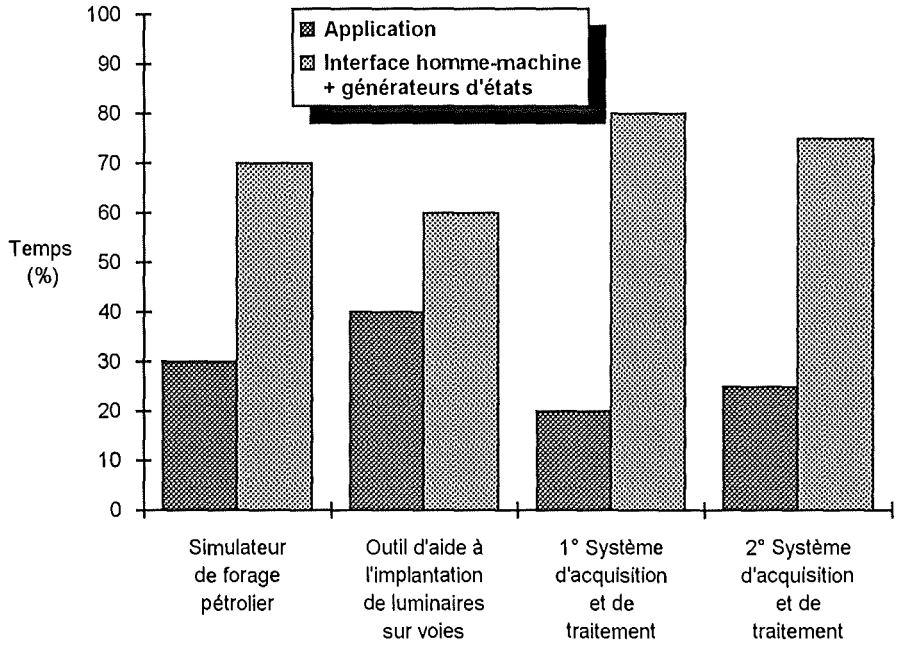
Le choix de ce mémoire m'a paru être un bon compromis entre, d'une part, l'apport pour la société d'outils qui lui sont nécessaires, et d'autre part, la mise en pratique d'une méthodologie de conception par objets comme support au développement d'Objets Interactifs (OIs).

Enfin, tout au long de ce mémoire, seuls les éléments essentiels concernant l'environnement Windows ou la méthodologie de conception seront détaillés.

I. Introduction.

a) Le contexte.

Après deux ans de développement au sein de *MTS* de logiciels techniques et scientifiques dans l'environnement *Windows*, nous pouvons faire un premier bilan sur la répartition du temps de développement entre l'interface homme-machine et le générateur d'états ou rapports d'une part et l'application d'autre part. L'application correspond à la sémantique du problème. Dans le monde SmallTalk, cette partie est définie par le Modèle et l'interface utilisateur par la Vue et le Contrôleur. Ces trois composants sont appelés dans la littérature MVC (Model View Controller) ou encore MPD (Model Pane Dispatcher).
On peut estimer que moins de 50 % du temps est consacré au développement de l'application[1].Ce chiffre se base sur une étude portant sur quatre logiciels développés par notre société qui ont une taille comprise entre 10.000 et 30.000 lignes de code. Nous avons repris ci-dessous la répartition de ces logiciels[2].



Ce graphique montre nettement que le temps de développement de l'interface utilisateur et du générateur d'états est prépondérant.

---

[1]Nous aurons également à l'avenir de plus en plus de demandes pour l'amélioration de la présentation d'anciennes applications (rewanping). Ce type de projet entraînera un déséquilibre encore plus marqué entre le temps de développement de l'application et de l'interface homme-machine.
[2]Etant donné la nature confidentielle de certains ces projets, nous ne pouvont les détailler.

La partie Interface utilisateur peut elle-même être subdivisée en deux parties, la première étant l'organisation du dialogue et la deuxième, la présentation à l'aide d'objets interactifs.

Organisation du dialogue:

Cette partie correspond au design des différents menus ainsi qu'à la gestion des différents items de menu. Cette architecture peut être définie par l'étude des fonctionnalités ou des objets de l'application. Dans l'environnement *Windows*, l'organisation est généralement orientée objet. Ainsi, au niveau du menu principal, on trouve des intitulés d'objets et, au niveau des sous-menus, on trouve la liste des opérations associées à chaque objet.

Présentation à l'aide d'objets interactifs:

Les boîtes de dialogue sont généralement constituées d'objets interactifs[3]. Elle sont développées avec des éditeurs de boîtes de dialogue. Ces outils graphiques permettent l'ajustement précis des différents objets interactifs. De plus, ils permettent de simuler le comportement de ces boîtes. La conception de ces différentes boîtes de dialogue peut être déduite de la découpe en objets de l'application. Notons également qu'à l'heure actuelle, on trouve sur le marché des outils de prototypages qui permettent de définir l'intégralité de l'interface utilisateur.

L'environnement *Windows* offre aux développeurs un ensemble d'objets interactifs qui couvrent une partie des besoins. Les objets interactifs *génériques* livrés en standard sont:

1. Button
2. Editbox
3. Listbox
4. Scrollbar
5. Static

Quoique couvrant un grand nombre de problèmes, il s'avère que cet ensemble est trop restreint.

Certains objets ne sont pas suffisamment spécialisés:

Si nous prenons l'exemple de la boîte d'édition, quoiqu'étant tout à fait générale, elle n'est pas adaptée pour faire des acquisitions sur des domaines de valeurs prédéfinis.

---

[3]Les objets interactifs sont en fait des objets de dialogue utilisés pour la saisie et/ou l'affichage d'informations.

Le seul recours est d'effectuer le contrôle de validité en-dehors de l'objet interactif. Cela entraîne un temps de réponse plus long, un contrôle à postériori et le mécanisme d'encapsulation est violé.

Certains objets font défauts:

Dans les systèmes d'informations, et en particulier ceux s'appuyant sur des moteurs relationnels, la notion de table est essentielle. La listbox n'est pas adaptée à la visualisation des tables. En effet, celle-ci n'est pas orientée multi-colonnes et ne permet aucune édition sur les éléments la constituant.
Dans les applications scientifiques il est bien souvent utile d'afficher des courbes. Cependant , il n'existe aucun objet interactif qui gère ce type de représentation.

Jusqu'il y a peu, pour pallier à ces inconvénients, nous développions des fonctions qui simulaient le comportement d'objets interactifs. Etant donné que ces différentes fonctionnalités n'étaient pas "*pensées*" sous forme d'objets interactifs, celles-ci ne s'intégraient pas dans l'environnement Windows et étaient rarement réutilisables si ce n'est en y apportant des modifications.
En effet, comme nous l'avons vu plus haut, les objets interactifs sont placés dans les boîtes de dialogue avec des éditeurs graphiques spécialisés. Comme nos "pseudo-objets interactifs" ne respectent pas tous les protocoles des objets interactifs, ils ne pouvaient être intégrés dans les boîtes de dialogue de façon interactive.

b) Les objectifs.

Vu les remarques ci-dessus, il nous a paru intéressant de nous doter d'une bibliothèque d'objets interactifs qui répondent à nos besoins. Celle-ci a été définie par l'équipe de développement. Ainsi dans cette bibliothèque nous trouverons les objets suivants :

Curve2D	Permet l'affichage de courbes, iso-courbes.
EditLong	Permet l'acquisition d'une valeur de type LONG.
EditFloat	Permet l'acquisition d'une valeur de type FLOAT.
EditDateTime	Permet l'acquisition d'une valeurde type DATE ou TIME.
EditMask	Permet l'acquisition d'une valeur contrainte par un masque de saisie.
GraphicButton	Equivalent de l'objet standard de type Button mais dont la représentation sémantique est graphique.
Meter	Permet de visualiser l'état d'avancement d'une tâche.
StateLine	Permet l'affichage d'informations dans une fenêtre principale.
Table	Permet l'affichage et l'édition d'une table.

La spécification des différents objets interactifs reprendra les services (interfaces) que ceux-ci doivent donner aux applications qui les mettront en oeuvre en respectant la philosophie de *Windows*. Certaines contraintes seront définies sur la gestion de l'information au sein même de l'objet interactif. De plus, ces différents objets interactifs devront être intégrés dans l'environnement *Windows* au niveau de l'éditeur de boîtes de dialogue[4].
Pour atteindre ces objectifs, le choix de la méthode de conception ne sera pas imposée, seule une contrainte au niveau de l'implémentation devra être respectée. Ainsi, le langage C (Microsoft C 6.0) sera le langage de programmation.

> *Note: Du point de vue du développeur, les objets interactifs sont en fait des composants réutilisables qui peuvent être vus comme des boîtes noires avec lesquelles on communique via des messages (interfaces). Certains auteurs les appellent également des "ICs" (Integrated Circuits) par analogie avec les circuits intégrés que l'on retrouve en électronique. Dans la littérature spécifique au développement d'applications sous Windows on retrouve souvent le terme "Control" comme définition d'un Objet Interactif.*

Vu l'ampleur du travail à réaliser, il semble raisonnable d'affirmer dès à présent que l'ensemble de la bibliothèque ne pourra être développée dans le seul cadre de ce mémoire.

Avant de spécifier de façon plus précise les objectifs du travail, il m'a semblé intéressant de définir l'environnement *MS Windows,* certains mécanismes qui seront utilisés pour la réalisation du travail ainsi qu'une présentation de la méthode de conception utilisée.

---

[4]Pour de plus amples informations sur l'éditeur de boîtes de dialogue se référer au manuel "Tools" livré avec la boîte à outils de Windows (SDK).

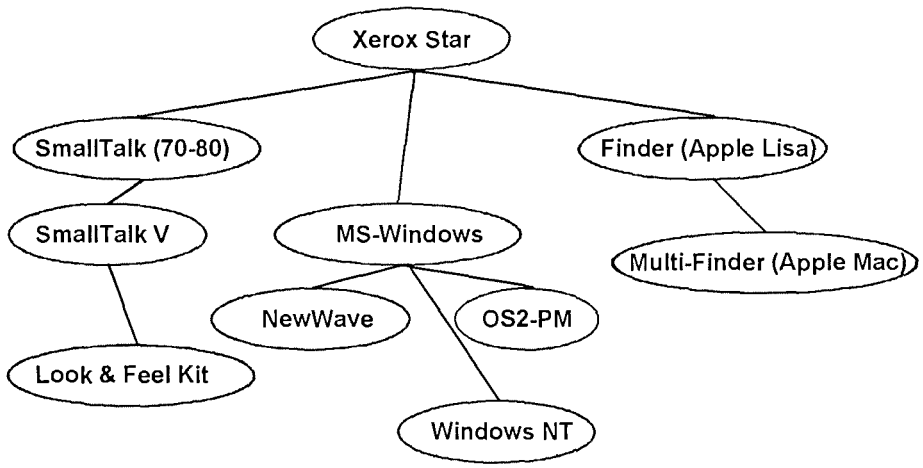
II. Environnement MS Windows.

a) Définition.

Comme première définition de l'environnement *Windows*, nous pouvons dire qu'il s'agit d'une interface utilisateur graphique (**G**raphical **U**ser **I**nterfaces). En fait, cette définition ne couvre qu'un aspect de cet environnement car celui-ci pourrait être assimilé à un système d'exploitation. Dans ce type d'environnement c'est l'interface qui dirige l'application. C'est un système événementiel.

b) Historique.

Dans les années 1970, un groupe de travail[5] du centre de recherche de Xerox Palo Alto se constitua pour définir une nouvelle façon dont les utilisateurs pourraient gérer l'information dans un environnement informatique. Ils ont travaillé sur deux grands axes: un langage de description qui servirait d'interface (intégrateur) entre le modèle du monde réel et celui du système d'information d'une part et d'autre part, un langage d'interaction entre l'utilisateur et le système d'information. Ainsi sont nés les concepts d'approche objets et d'interface utilisateur graphique. Ces concepts ont été concrétisés par un environnement de développement nommé SmallTalk (SmallTalk 70 puis 80).
Ils ont été repris par la société Apple qui a créé l'ordinateur LISA puis le Macintosh en 1984.
Suite au succès rencontré par la société Apple, la société *Microsoft* Corporation annonça *Windows* en 1983 et la première version fut disponible dès 1985. Mais le grand démarrage de cet environnement fut l'annonce de la version 3.0 dès 1989 et sa venue sur le marché durant l'année 1990.
Depuis, la plupart des sociétés qui éditaient des logiciels pour l'environnement Dos n'ont pas cessé de porter ceux-ci dans l'environnement *Windows*.



[5]The Xerox Palo Alto Research Center Learning Research Group puis renommé the Xerox Palo Alto Research Center Software Concepts Group en 1981.

c) Fonctions de cet environnement.

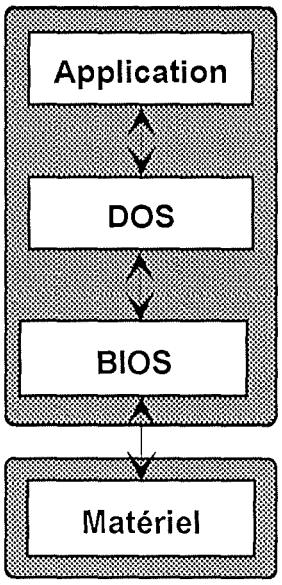
Par rapport à la définition que nous avons donné précédemment, nous allons détailler les différentes fonctionnalités d'un tel environnement.

1. Indépendance vis-à-vis du matériel.

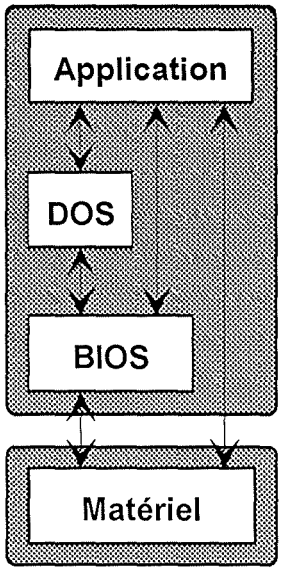
Un ordinateur de type PC est composé d'un ensemble de matériels: unité centrale, mémoire, clavier, écran, disques ... qui sont gérés par des circuits spécialisés. La communication entre le logiciel et ces circuits se fait de plusieurs façons: interruptions provoquées par un matériel et générant un appel de fonction à une adresse dépendant du type d'interruption, "mapping" sur des adresses particulières en mémoire et utilisation de ports d'entrée-sortie.

Pour éviter que les logiciels ne soient liés à un ordinateur particulier, une couche intermédiaire le BIOS est introduite entre le matériel et le logiciel. Elle offre des services aux logiciels par des interruptions logicielles.

Il existe une couche supérieure qui s'appelle le DOS qui offre des services de plus haut niveau (gestion des disques, répertoires et fichiers).

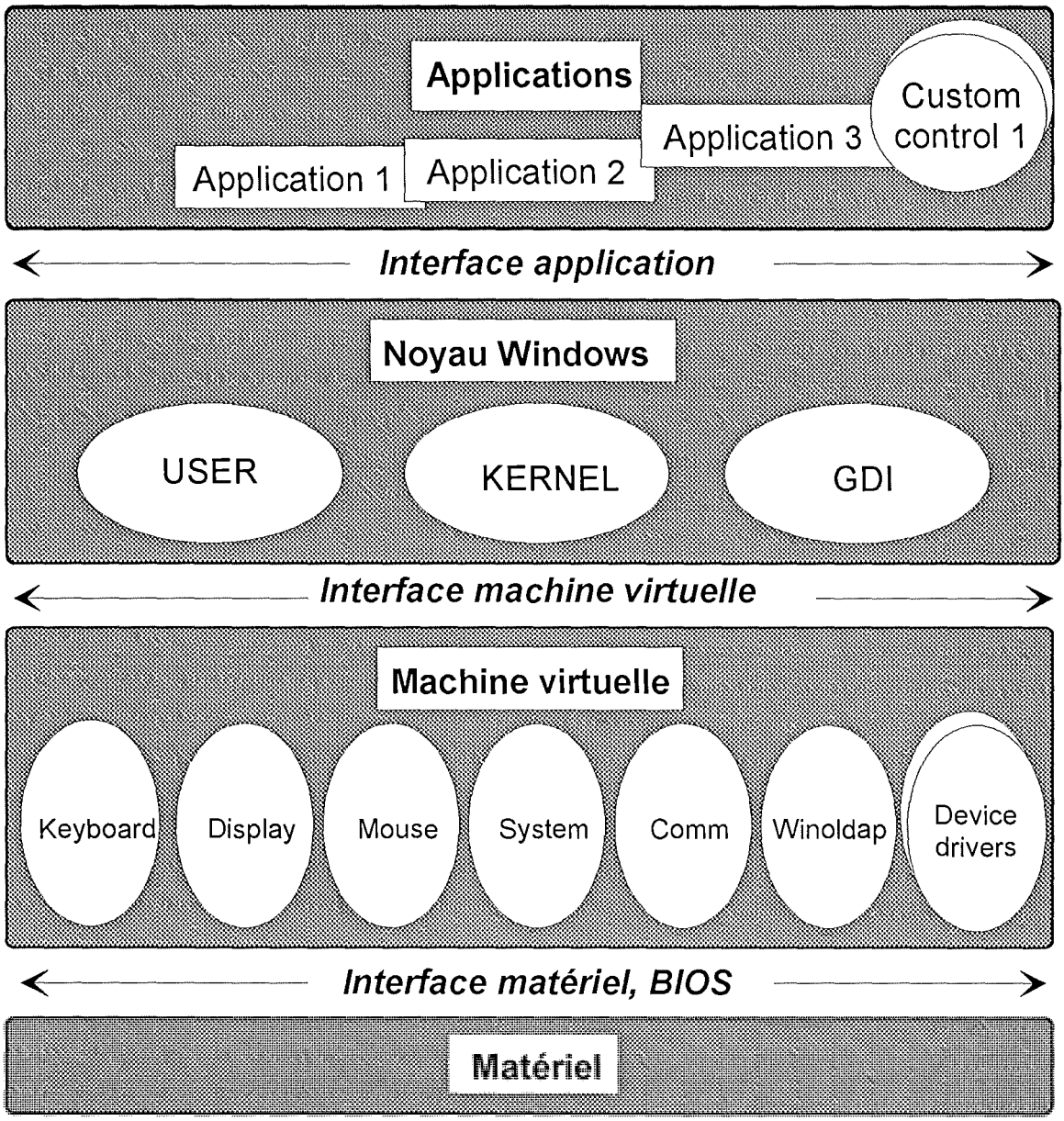


Mais beaucoup d'applications sont construites à partir des services offerts par la couche BIOS et même à partir du matériel et cela pour améliorer les performances ou pour gérer du matériel spécialisé.



MS Windows a été conçu pour éviter au programmeur de se soucier du type de matériel sur lequel s'exécute son application. Tous les périphériques d'un micro-ordinateur sont virtualisés: clavier, écran, souris, imprimante, table traçante, port de communication asynchrone. Ces périphériques virtuels sont appelés des pilotes (drivers) qui fournissent une interface standard indépendante du matériel mis en place.

MS Windows est constitué de deux couches fonctionnelles: le noyau qui est indépendant de tout matériel, et une machine virtuelle réalisant l'interface avec le matériel[6].



[6]A noter que l'on retrouve le principe de machine virtuelle dans l'environnement SmallTalk. Celle-ci est plus générale que la machine virtuelle de MS *Windows*.

- *Le noyau.*

C'est un ensemble de logiciels qui constitue le moteur du système qui est indépendant du matériel installé et ainsi est toujours le même.

Il comprend 3 modules:

KERNEL gère l'allocation des ressources telles que la mémoire; il charge les applications (chargeur) et réalise le multi-tâche (scheduling).

USER comprend le gestionnaire de fenêtres. Il est chargé de créer et de maintenir les fenêtres à l'écran, de les déplacer, de changer leur taille, de définir leur superposition. Ce module reçoit également les informations des moyens d'entrée (clavier, souris, stylo, ...) et les diffuse aux fenêtres destinataires.

GDI (Graphics Device Interface) contient la bibliothèque de primitives graphiques et permet de créer des images sur un périphérique (écran, imprimante ...)

Les applications accèdent aux services du noyau à travers trois interfaces:

System Services Interface (KERNEL)
Windows Manager Interface (USER)
Graphics Device Interface (GDI)

En fait, au niveau du développeur, il n'y a pas de distinction. Celui-ci dispose d'un ensemble de fonctions disponibles dans le "Software Development Kit" (SDK) qui lui permette d'accéder à tous les services disponibles de *MS Windows.*

- *La Machine Virtuelle.*

Elle est composée d'un ensemble de pilotes (drivers) gérant les ressources matérielles dépendantes du système physique. Chacun des pilotes offre des services au noyau.

La Machine Virtuelle est composée de plusieurs modules:

KEYBOARD est le module de gestion clavier et du haut parleur.

MOUSE est le module chargé de détecter les événements générés par la souris (déplacement, appui sur le bouton gauche ...) et de les transmettre au noyau.

DISPLAY est le module qui traite les sorties vers l'écran. Il gère également la position du curseur.

SYSTEM est le module de gestion de l'horloge et des disques. En fait, *Windows* n'a qu'une responsabilité limitée dans la gestion des disques, le reste étant du ressort du DOS.

COMM contrôle les communications à travers les ports série(s) et parallèle(s).

SOUND génère les sons sur le haut parleur.

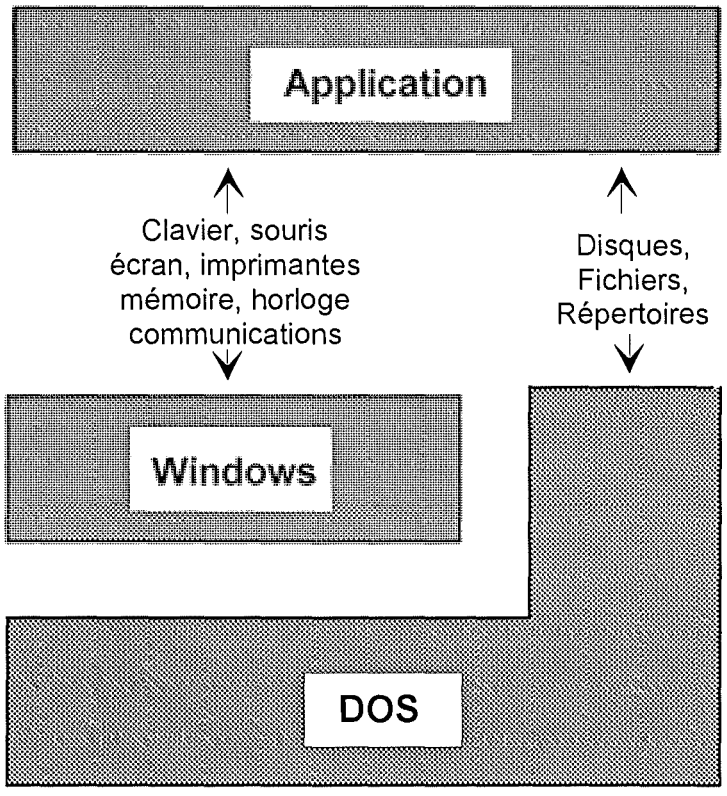
WINOLDAP est un module qui gère la préparation du chargement d'applications non *Windows* et en permettant l'exécution.

NETWORK (option) est un module d'interfaçage entre l'environnement MS *Windows* et le réseau local si celui-ci est installé.

Il existe pour chaque périphériques de type imprimante, table traçante, souris, ... des pilotes (Device Drivers).

Cette machine virtuelle n'est pas accessible au développeur. A noter, que l'ajout de nouveaux pilotes se fait grâce au "Device Development Kit" (DDK).

Enfin, on peut remarquer que Windows n'a pas de pilotes pour gérer les fichiers. Ainsi, les primitives du DOS pour l'accès aux fichiers sont nécessaires.



2. Le Multi-tâche.

Windows offre également le multi-tâche non-préemptif.
Windows gère le partage des différentes ressources (écran, clavier, souris, timer, mémoire) entre les différents programmes.
Dans un système multi-tâche non-préemptif, un programme ne peut être interrompu par le système d'exploitation. De ce fait, un programme doit donner la main au système d'exploitation pour que celui-ci puisse exécuter d'autres programmes.

3. La gestion de la mémoire et le concept de DLL.

Avec l'arrivée de la version 3.0 de *MS Windows* et les ordinateurs basés sur des processeurs de type 80286, 80386 et 80486, la gestion de la mémoire a été améliorée. En effet, suivant le type de processeur, la gestion de la mémoire diffère pour améliorer les performances de l'environnement. Le but n'étant pas de détailler la gestion de la mémoire sous Windows, nous allons nous concentrer sur les problèmes de gestion mémoire lors de l'utilisation de DLLs (Dynamic Link Libraries). En effet, nos différents objets interactifs seront accessibles via ce mécanisme.

- *Editeur de lien statique.*

  Lors de l'opération d'édition de lien en mode statique, le code et les données des fonctions qui sont dans des librairies sont copiés dans le programme exécutable. Ainsi si cette fonction est modifiée, il faut refaire l'opération d'édition des liens. De plus, le code de l'exécutable est relativement grand. Enfin si cette fonction est partagée par plusieurs applications (tâches), elle devra être copiée dans chaque exécutable.

- *Editeur de lien dynamique.*

  A l'inverse du précédent , en mode dynamique, le code et les données des fonctions ne sont pas copiés dans l'exécutable. Seules des informations d'accès à ces fonctions y sont inscrites. Ainsi, le code est plus court, la modification d'une fonction ne doit pas entraîner une nouvelle opération d'édition des liens et, si ces fonctions sont accessibles par plusieurs programmes (tâches) leur code ne sera présent en mémoire q'une seule fois.
  Au niveau de la gestion mémoire il faut prendre certaines précautions.
  Lors de l'appel d'une fonction située dans une dynamic link library (DLL) par un exécutable, le Data Segment (DS) est chargé avec l'adresse du Data Segment de l'appelant. A l'entrée dans la DLL, il faut sauvegarder le DS de l'exécutable et charger celui de la DLL. En retour, il faudra recharger le DS de l'exécutable. Ces opérations sont effectuées par le prolog et l'épilog insérés en début et en fin de fonction FAR (adresse sur 32 bits, dont 16 pour le segment) par des switchs de compilation.

De plus, étant donné que les fonctions de la DLL sont appelées depuis un segment de code différent du leur, elles doivent être déclarées en FAR.
Lorsqu'une fonction utilise un pointeur sur une donnée qui n'est pas située dans son Data Segment, elle doit utiliser un Far Pointer qui va comporter l'adresse du segment où cette donnée est situé. En particulier, les données dans le stack n'appartiennent pas au Data Segment de la DLL puisque c'est le stack du programme appelant. Tous les pointeurs passés en paramètres à une DLL doivent être des pointeurs de type FAR parce qu'ils pointent en général sur des données situées dans le Data Segment de l'appelant. De même, lorsqu'une DLL utilise un pointeur sur sur une donnée située dans le stack, elle doit également utiliser un pointeur FAR car le Stack Segment (SS) n'est pas identique à son Data Segment (DS).

4. La communication inter-applications.

Cette communication est réalisée par les moyens suivants:

- *Presse-papier (Clipboard).*

  Le presse-papier est une zone mémoire partagée entre toutes les applications tournant sous *Windows* qui permet d'échanger des informations entre celles-ci. En standard, il existe cinq types de données qui sont reconnus par le protocole du presse-papier. Cependant, il est possible de définir d'autres types de données: il suffit pour cela que l'émetteur et le récepteur s'entendent sur le format du nouveau type de données.

- *Protocole DDE (Dynamic Data Exchange).*

  Le DDE est un protocole de communication permettant à des applications n'ayant aucun lien entre elles d'échanger des données de manière dynamique. Ce protocole s'effectue par échange de messages entre applications puis échange de données par l'intermédiaire de la mémoire globale.
  Dans ce protocole on retrouve la notion de client-serveur.

- *Protocole OLE (Object Linkage Embeded).*

  Ce type de protocole apparaît dans la version 3.1 de *Windows*. Il peut être assimilé à celui du clipboard avec des extensions. Avec ce dernier, si une donnée est copiée d'un document source vers un document destination et si on modifie la donnée dans le document source, il n'y aura pas de répercussion dans le document destinataire. Avec le protocole OLE, la répercussion peut se faire si on définit un lien sur la donnée. De plus, on peut éditer la donnée dans le document destinataire. En effet, un double-clic souris sur la donnée active l'application qui l'a générée.

5. *Une interface utilisateur standardisée.*

- *Du point de vue l'utilisateur.*

  Les applications qui sont développées suivant certaines règles ergonomiques dans ce type d'environnement, apportent à l'utilisateur final des facilités d'utilisations considérables. Ainsi, les fonctionnalités qui sont communes à toutes les applications seront définies partout de la même façon. Cette définition porte sur l'organisation spatiale des objets interactifs, menus et items de menus, les intitulés, les raccourcis et accélérateurs clavier de ces derniers.
  Au niveau de l'interface utilisateur, il existe la norme définie par IBM: **C**ommon **U**ser **A**ccess (CUA) qui fait partie du **S**ystems **A**pplication **A**rchitecture (SAA).
  De plus, l'utilisateur pourra échanger des informations entre les différentes applications via les protocoles d'échanges.
  Enfin, ce type d'environnement offre certaines garanties quant à l'indépendance de son application vis-à-vis du matériel utilisé.

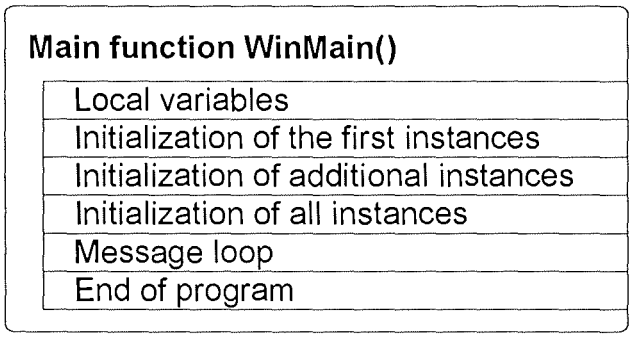
- *Du point de vue du développeur.*

  Ce type d'environnement offre au développeur un ensemble d'outils et protocoles qui vont lui permettre de produire des applications standardisées.
  De plus, l'environnement lui offre une machine virtuelle qui le rend indépendant du matériel uitlisé.

d) Principes de fonctionnement de Windows.

Les deux principes de base sur lesquels toutes les applications sont développées pour l'environnement MS *Windows* sont: les *fenêtres* et les *messages*.
En effet, la plupart des applications qui tournent dans cet environnement ont au moins une fenêtre qui leur est associée.

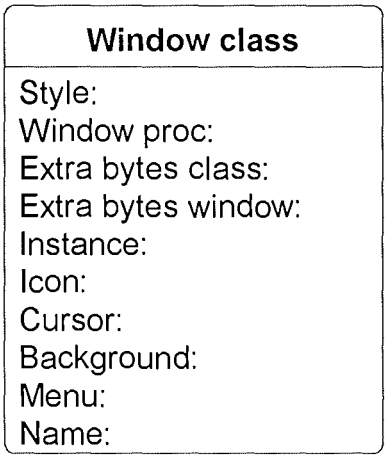
La structure d'un programme est la suivante:



- La fonction principale WinMain() est similaire à la function main()[7] du langage C.
- L'initialisation de la première instance permet d'enregistrer la classe. Cette opération "fait connaître" la classe de fenêtre définie pour cette application à l'environnement *Window*. Elle permet également de charger des ressources.
- L'initialisation des instances supplémentaires permet par exemple d'incrémenter le compteur d'instances de cette application.
- L'initialisation de toutes les instances est la phase de création de fenêtres.
- La boucle des messages est en fait le coeur de l'application par où vont transiter tous les messages.
- La phase de terminaison permet de libérer la mémoire allouée dynamiquement.

[7]Main() indique qu'il s'agit d'un programme principal et non d'une fonction qui serait appelée par le programme principal. Cette fonction est utile pour passer des arguments.

La structure d'une classe de fenêtre est la suivante:



- La variable "style" permet de spécialiser l'aspect visuel et/ou le comportement de la fenêtre.
- La variable "Window proc" est une référence (pointeur) sur la fonction qui définit le comportement de la fenêtre.
- La variable "Extra bytes class" peut être assimilée à une variable de classe présente dans les langages orientés objets.
- La variable "Extra bytes window" peut être assimilée à une variable d'instance présente dans les langages orientés objets.
- La variable 'Instance" stocke la référence du process qui à créé et enregistré la classe.
- La variable "Icon" stocke la référence à l'icône qui sera affichée lorsque la fenêtre sera minimisée.
- La variable "Cursor" stocke la référence sur le modèle du curseur qui devra apparaître lorsque celui-ci sera présent dans l'espace client de la fenêtre.
- La variable "Background" stocke la référence sur le modèle du fond de fenêtre.
- La variable "Menu" stocke la référence sur un menu si cela est nécessaire.
- La variable "Name" stocke le nom de la classe.

La structure d'une fonction associée à une classe de fenêtre est la suivante:

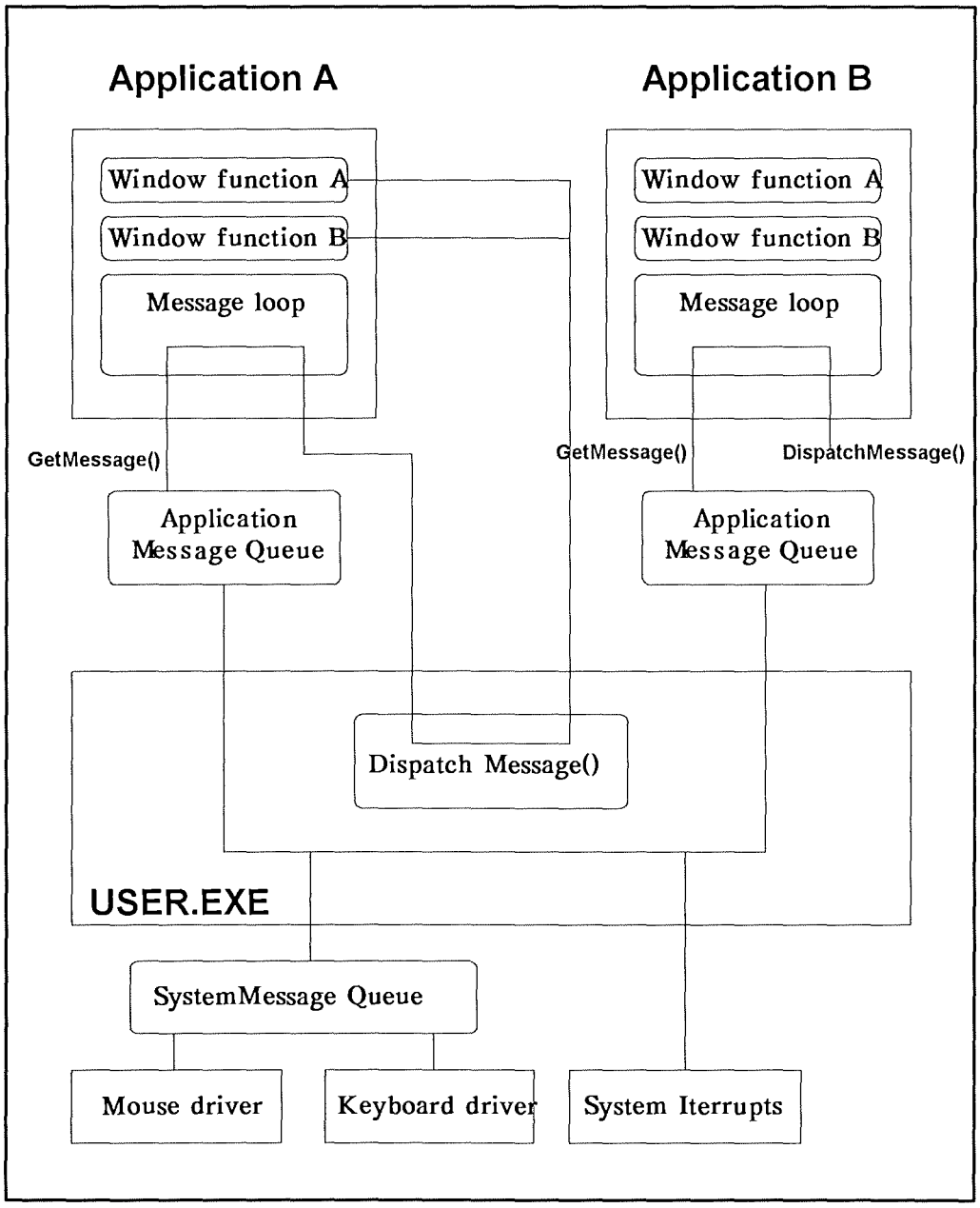
```
LONG FAR PASCAL MainWndProc(HWND hWnd, unsigned msg, WORD wParam, LONG lParam)
{
 switch (msg)
  {
   case WM_X:
        ...
        break;

   case WM_Y:
        ...
        break;

   case MM_Z:
        ...
        break;

   default:
        return (DefWindowProc(hWnd, msg, wParam, lParam);
  }
 return 0L;
}
```

Cette fonction est de type "Call back". En effet, elle sera appelée par l'environnement *Windows* si un message doit être traité par celle-ci.
Les messages qui sont préfixés par "WM_" sont des messages prédéfinis dans l'environnement Windows. Seuls ceux qui seront traités de façon spécifique seront repris. Les messages prédéfinis qui ne doivent pas être traités seront passés à la fonction "DefWindowProc(...)". Les messages préfixés par le "MM_" sont des messages définis pour la classe de fenêtre spécifique.

Notons également que le message a un format bien défini.
Celui-ci est de la forme: SendMessage(hWnd, WM_MESSAGE, wParam, lParam).

- Le paramètre "hWnd" est la fenêtre réceptrice du message.
- Le paramètre "WM_message" est le message (variable).
- Les paramètres wParam et lParam qui sont respectivement de type WORD et LONG sont dépendants du type de message.

Nous allons voir maintenant comment l'application est prise en charge par l'environnement.



La file des messages système est utilisée pour la communication entre les différentes fenêtres ou une fenêtre et l'environnement *Windows*. C'est l'environnement qui dispatche les messages provenant de la file système vers les files des applications via la fonction "DispatchMessage()".
La file d'une application est gerée par la fonction "GetMessage()" en FIFO. Si elle est vide (ou si un message pour une autre application a une priorité plus élevée), l'environnement *Windows* passe le contrôle à une autre application (à l'autre). Sinon, l'environnement fait appel à la fonction associée à la fenêtre pour traiter le message.

e) Limitations et extensions.

Au niveau des limitations, nous nous bornerons à celles qui sont liées à l'objet de ce travail.
Comme nous l'avons déjà vu précédemment, une des limitations se situe au niveau de la spécialisation des objets interactifs et de leurs diversités.
Cependant, nous voyons arriver sur le marché des objets interactifs développés par des sociétés de service. Malheureusement, seuls des objets interactifs susceptibles d'être vendus en grand nombre sont proposés. Ainsi, il n'existe pas d'OIs pour supporter l'affichage de courbes et surfaces.

Quant aux extensions des OIs standards, elles sont parfois hasardeuses. En effet, dans la littérature spécialisée, on déconseille fortement la spécialisation d'OI tel que le bouton. Un recours possible est de définir cet objet "Owner draw". Mais, par ce mécanisme, l'encapsulation est violée. La technique "Owner draw" consiste en fait à déléguer certaines opérations à l'application (fenêtre parent). Dans l'exemple du bouton de type "Owner draw"", lors de l'affichage de son contenu (sélectionné ou pas), il envoie un message à la fenêtre parent pour lui demander de l'afficher. Par cette technique, le comportement est partagé entre l'objet et l'application.

e) Avantages et Inconvénients.

1. Avantages.

- Comme nous l'avons déja vu, grâce à la machine virtuelle, les applications sont indépendantes du type de matériels utilisés.

- En respectant certaines règles de programmation, il est possible de rendre l'application (modèle) indépendante de l'interface utilisateur.

- Par le principe événementiel et en respectant certaines contraintes, c'est l'interface utilisateur qui dirrige l'application.

- Comme l'environnement *Windows* est graphique il est possible de diminuer le golfe d'exécution et d'évaluation. Cela est rendu possible en utilisant des objets interactifs qui ont une représentation graphique proche du monde de l'utilisateur et qui supportent des actions qui lui sont familières.

- Grâce à une certaine normalisation des différents environnements et/ou des applications, l'utilisateur travaille dans un environnement homogène.

- La communication inter-applications offre à l'utilisateur une nouvelle dimension.

- Les outils qui sont offerts aux développeurs sont puissants et variés.

- Le code des applications peut atteindre une taille importante (Il n'y a plus de limitation sur la taille du code).

2) Inconvénients.

- Les inconvénients liés à ce type d'environnement concernent surtout les développeurs. Le temps d'apprentissage et de maîtrise des outils et concepts de développement est élevé. Certaines sociétés estiment que le temps de maîtrise du système de développement de *Windows* (SDK) peut dépasser six mois.

- Dans une moindre mesure, on peut être confronté à un problème de performance. En effet, ce type d'environnement demande des ressources matérielles importantes.

- Par l'asynchronisme, le système peut être mal adapté pour les systèmes d'acquisitions.

III. Méthode utilisée pour la conception des composants.

Avant de définir le choix de la méthode qui sera utilisée pour spécifier les différents objets interactifs, nous allons tenter de positionner les concepts présents dans *Windows.*
Ainsi, dans cet environnement, nous retrouvons certains concepts présents dans les langages et méthodes orientés objets.

Ces concepts sont: La classe définit le comportement de toute instances dérivées de celle-ci.
L'instance est créée par instanciation d'une classe.
Le message est envoyé à une instance d'une classe.
Le polymorphisme est présent.

Par contre, le concept d'héritage n'apparaît pas directement. Mais si on se réfère à la structure d'une fonction associée à une classe de fenêtre on remarquera la présence de l'appel à la fonction "DefWindowProc(...)" qui définit le comportement par défaut de toute fenêtre. Enfin, nous verrons par la suite que ce mécanisme peut être appliqué de façon récursive.
De plus, les objets interactifs peuvent être vus comme des boîtes noires ou des "Integrated Circuits " (ICs) qui offrent certains services.
Ainsi, il nous a paru intéressant d'utiliser une méthode orientée objets pour spécifier les différents composants.

Maintenant que le choix du type de méthode est défini, il nous reste à choisir entre une méthode qui soit basée sur une démarche rigoureuse et stricte ou sur une approche par prototypage. Ces dernières ont généralement des modèles bien définis mais la démarche est surtout basée sur une approche par essais et erreurs (itérative). Pour pratiquer ce genre d'approche, il nous semble que des outils hautements interactifs soient indispensables. Ainsi des environnements tels que SmallTalk sont adaptés pour supporter ce type d'approche.
De plus, lors d'une approche par prototypage, une reformalisation est toujours nécessaire après la conception pour permettre une maintenance et une évolutivité plus facile.
Ainsi, le choix d'une méthode basée sur une démarche stricte nous paraît adéquate. Celle-ci devra être suffisamment générale (réutilisabilité).
Nous n'allons pas reprendre ici l'ensemble des différentes méthodes de conception par objets que nous trouvons dans la littérature, mais nous allons justifier notre choix de la méthode appellée: "Object-Oriented Modeling and Design" (OMT)[8].

---

[8]Object-Oriented Modeling and Design (OMT) par J.Rumbaugh, M Blaha, W Premerlani, F Eddy, W Loresen (General Electric Research and Development Center).

Cette méthode repose sur 3 modèles de base:

- Modèle de la statique: objet,
- Modèle de la dynamique: états-transitions,
- Modèle fonctionnel: diagramme des flux.

Ces modèles sont relativement proches des modèles présents dans la méthode d'analyse vue au cours de conception de systèmes d'informations de la 1ère licence & maîtrise en informatique aux F.U.N.D.P.

Cette méthode se base sur 3 phases de développement qui sont:

- Analyse,
- Design du système,
- Design des classes.

Ces différentes phases correspondent assez bien avec la démarche qui sera nécessaire dans la conception des différents composants. En effet, n'oublions pas que nous devons respecter la "philosophie *Windows*" qui est relativement restrictive à certains points de vue (arguments des messages, organisation des différents modules).

a) Définition des différents modèles OMT.

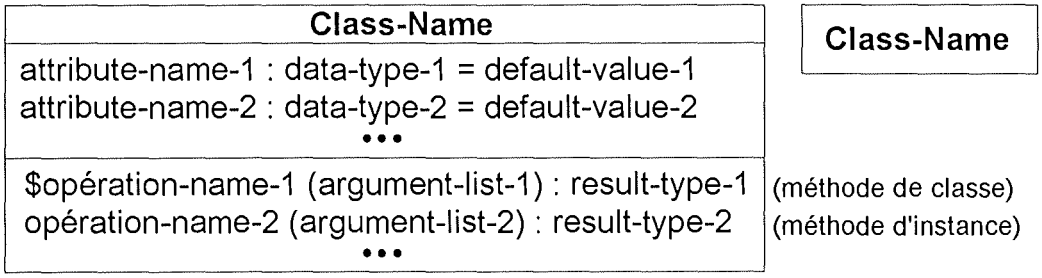
Les 3 modèles sont représentés par un formalisme graphique

1. Modèle de la Statique: Objet

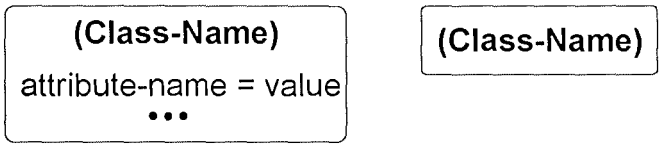
Ce modèle décrit la structure des classes dans le système, c'est à dire leurs identités, les relations entre elles, leurs attributs et leurs opérations (leurs comportements). Ce modèle va nous donner une représentation formelle de l'ensemble des objets du monde réel de l'application.

Différence entre Classe d'objets et Objet :	*Les objets sont tous distinguables, mais certains objets appartiennent à une même classe qui définit le comportement de ceux-ci.*

- *Diagrammes des Classes.*

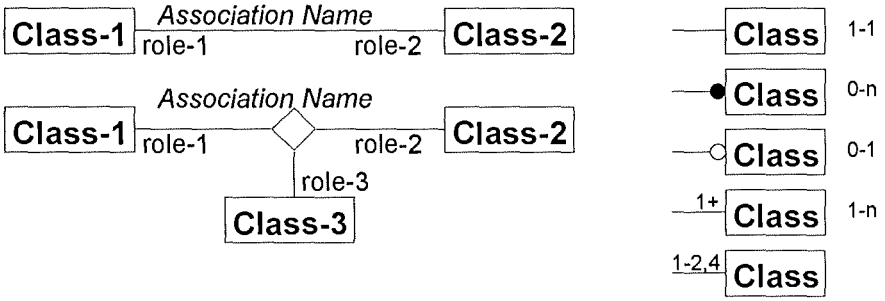


- *Diagramme des Objets (Instances).*

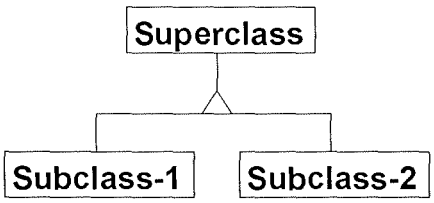


- *Mode de représentation des relations.*

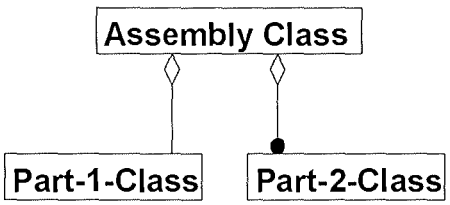
Relation d'association :



Relation de généralisation (héritage) :



Relation d'aggrégation :



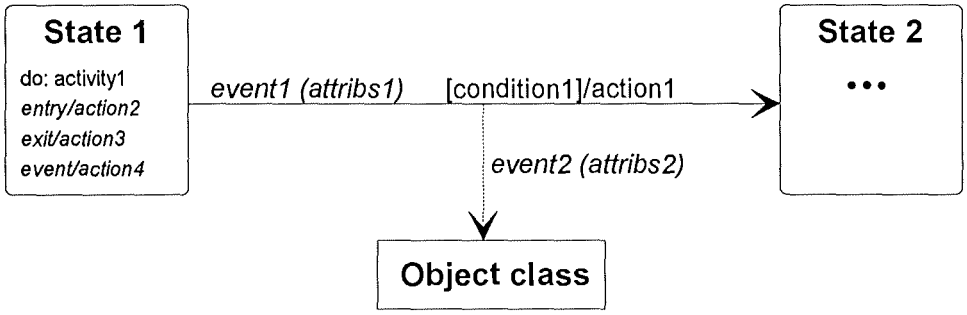
Relation d'instanciation

Notons que les différents diagrammes ci-dessus ne représentent qu'une partie de la sémantique de la méthode.

2. *Modèle de la Dynamique.*

Ce modèle décrit l'évolution des objets et leurs relations en fonction du temps. Ainsi, ce modèle décrit la séquence des opérations d'un objet sans se préoccuper du comment. Ce modèle est basé sur la théorie des états-transitions.
Pour chaque classe d'objets, nous aurons un modèle de la dynamique. La communication entre classes se fera via un envoi d'événements (messages) .

- *Diagramme d'états.*



- *Définitions.*

  Activity: une opération qui prend un certain temps.

  Action: une opération qui est instantanée.

  Event: Stimulus envoyé à un objet (instantané).

  State: une abstraction des attributs et/ou liaisons d'un objet

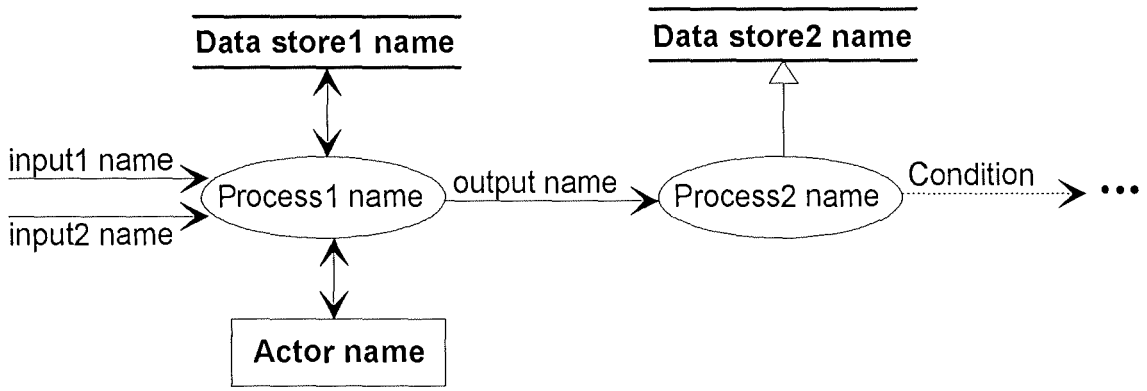
- *Principe.*

  L'objet passe de l'état 1 à l'état 2 s'il reçoit l'évènement 1 et que la condition 1 est respectée. Cet événement peut comporter des arguments 1, déclencher une action 1 et envoyer un événememement à un objet.
  Lorsque l'objet est dans l'état 2 il va entreprendre l'activité qui est associée à cet état. Si, pour tous les événements qui font passer l'objet dans cet état, leur action est identique alors on peut associer à l'état cette action en entrée. De même, pour tous les événements qui font sortir l'objet de cet état, leur action est identique alors on peut associer à l'état cette action en sortie. Si un événement ne provoque pas de changemement d'état on peut associer la paire événement-action à l'état. Enfin, un événement peut déclencher un évènement vers un autre objet.

3. Modèle Fonctionnel.

Le modèle fonctionnel décrit les opérations. Celui-ci montre comment les valeurs de sortie sont dérivées des valeurs d'entrée et des ressources sans se préocuper de l'ordre des opérations. Ainsi, pour chacune des opérations définies au niveau du modèle objet et/ou pour chaque action du modèle de la dynamique, nous trouverons un diagramme fonctionnel.

- *Diagramme du modèle fonctionnel.*



- *Définitions.*

  Actor: objet actif qui peut produire ou consommer des ressources

  Data store: objet passif de stockage.

- *Principe.*

  Un processus reçoit en entrée des informations et produit en sortie des informations. Il peut accéder à des ressources en lecture ou en écriture. Un processus peut définir un objet qui sera la cible d'un autre processus (Sélection d'une valeur parmis un ensemble). Ce mécanisme est représenté graphiquement par : ⇨

b) Méthodologie.

1. Phase d'analyse.

Cette phase sert à modéliser l'application et à comprendre le domaine sur lequel elle opère. Comme point de départ de cette phase nous avons l'expression des besoins. Celles-ci seront enrichies par des interviews avec les demandeurs et par les connaissances acquises sur le domaine. Comme résultat produit par cette phase nous aurons les différents modèles décrits précédemment.

2. Phase du design du système.

L'architecture globale du système est définie par cette phase. Ainsi, il sera organisé en sous-systèmes qui seront composés d'objets (classes). Cette organisation peut dépendre :

- de la communication inter-processus,
- du stockage des informations,
- de l'implémentation.

Dans cette phase, on ne modifie en rien l'architecture d'un objet. On procède à un groupement d'objets en sous-systèmes suivant certains critères.
Enfin, nous trouvons plusieurs modes d'organisation ou de découpage: en couche ou en partition.

3. Phase du design des classes.

Durant cette phase, les différents modèles sont raffinés de façon à optimiser les performances.
Alors que pendant la phase d'analyse on ne raisonne qu'en termes de l'application, ici on raisonne en termes informatiques.
On détermine les algorithmes et les structures qui seront utilisés pour l'implémentation.
Ici, on peut éventuellement modifier les modèles d'un objet (classe) afin de tenir compte de problèmes tels que: les performances, la place mémoire.

IV. Spécification des Objets interactifs.

a) Démarche pour spécifier les objets interactifs.

Avant de spécifier les différents objets interactifs, nous allons présenter à l'aide d'un exemple la façon dont nous allons définir le comportement d'un objet interactif[9]. L'objet choisi sera la "List Box" qui est intégrée dans l'environnement.
Certains types d'objets interactifs peuvent être spécialisés via des attributs de style. Ceux-ci peuvent aussi bien définir un aspect graphique particulier qu'un comportement spécifique. Ces attributs sont définis lors de la création d'une instance de la classe.
Au niveau du comportement, on peut considérer qu'il existe trois types de messages associés à un objet interactif.
Le premier type de messages associé à l'OI est celui qu'une application peut lui envoyer. Ce sont les messages de contrôle
Le second type de messages est celui que l'objet lui-même peut envoyer à l'application suite à la modification de son état. Ce sont des messages de notification.
Le troisième type de messages n'est pas connu de l'application; il correspond à des envois de messages à lui-même pour sa gestion propre. Ce sont les messages de commandes. Dans certains langages orientés objets on les appellerait des messages privés.
Dans l'exemple qui va suivre nous ne détaillerons que les messages des deux premiers types. En effet, seuls ces messages sont connus de l'application (du développeur qui utilise l'objet interactif).
Nous terminerons la spécification par la liste des actions que l'utilisateur peut effectuer sur ce type d'objet.

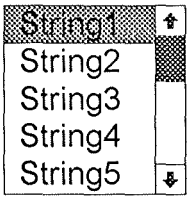
L'objet interactif LIST BOX:

La liste box est une fenêtre dans laquelle on peut placer des chaînes de caractères en standard ou tout autre objet graphique en mode "Owner draw" sous forme de colonnes. L'utilisateur a la possibilité de choisir un ou plusieurs items (dépendant du type). Si le nombre d'items est plus élevé que ceux visualisés, une barre de défilement vertical qui permet de "browser" l'ensemble des items s'ajoute à la fenêtre.

---

[9]En effet, certains auteurs estiment que le manuel d'utilisation d'une application peut être un bon point de départ pour définir le cahier des charges.

1. Repésentation graphique.



List box standard

2. Attributs de spécialisation de l'objet.

LBS_EXTENDSEL	La list box est à sélection multiple. La sélection se fait via la souris et la touche SHIFT ou CONTROL.
LBS_HASSTRINGS	Les items de la liste Owner-draw sont des chaînes de caractères.
LBS_MULTICOLUMN	La liste peut avoir plusieurs colonnes et défiler horizontalement.
LBS_MULTIPLESEL	La list box est à sélection multiple.
LBS_NOINTEGRALHEIGHT	La hauteur de l'objet n'est pas ajustée pour afficher un nombre entiers d'items.
LBS_NOREDRAW	La list box n'est pas redessinée à chaque changement.
LBS_NOTIFY	La fenêtre mère reçoit un message de notification à chaque clic de souris simple ou double.
LBS_OWNERDRAWFIXED	L'application doit dessiner les items ayant tous la même hauteur.
LBS_OWNERDRAWVARIABLE	L'application doit dessiner les items ayant des hauteurs différentes.
LBS_SORT	Les éléments de la liste sont triés.
LBS_USETABSTOPS	Utilise les tabulations pour le texte.
LBS_WANTKEYBOARDINPUT	La fenêtre parent reçoit les messages WM_VKEYTOITEM et WM_CHARTOITEM à chaque appui sur une touche.

De plus, certains styles sont hérités de la classe Window tels que: WS_BORDER, WS_HSCROLL, WS_VSCROLL.

3. Messages de contrôle.

```
lCode = SendMessage(hWnd, LB_ADDSTRING, 0, (LONG)(LPSTR)szString)
```

Ce message permet d'ajouter une chaîne de caractères dans la liste. Si la liste est définie comme non triée, elle sera ajoutée à la fin, sinon elle y sera insérée. De plus, si une chaîne était sélectionnée, elle sera désélectionnée.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur une chaîne qui doit être terminée par un caractère de fin de chaîne '\0',
- Le retour du message est l'index qui correspond à la position de la chaîne dans la liste ou un code d'erreur si un problème est survenu.

```
lCode = SendMessage(hWnd, LB_DELETESTRING,  wIndex, 0L)
```

Ce message permet de supprimer la chaîne située à l'index donné.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est l'index de la chaîne à supprimer,
- Le quatrième n'est pas utilisé,
- Le retour du message est le nombre de chaînes présentes ou un code d'erreur si un problème est survenu.

```
lCode = SendMessage(hWnd, LB_DIR,  wAttributes, (LONG)(LPSTR)szFilter)
```

Ce message permet d'ajouter une liste de nom de fichiers à partir du répertoire courant.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit les attributs des fichiers qui doivent être ajoutés dans la liste. Comme attributs nous avons: fichiers en lecture & écriture, fichiers en lecture, fichiers cachés, fichiers systèmes, sous-répertoires, disques. Ces attributs peuvent être cumulés.
- Le quatrième définit un filtre pour le nom des fichiers. Comme filtre nous pouvons avoir : "*.*" (sélection de tous les fichiers), "*.TXT" (sélection des fichiers ayant "TXT" comme extension), ...

- Le retour du message est le nombre de chaînes chargées ou un code d'erreur si un problème est survenu.

```
lCode = SendMessage(hWnd, LB_FINDSTRING, (WORD)nIndex, (LONG)(LPSTR)szPrefix)
```

Ce message permet de retourner l'index de la première chaîne qui a le préfixe défini par lParam.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit l'index à partir duquel il faut effectuer la recherche. Si la valeur est égale à -1, la recherche s'effectue au début,
- Le quatrième est un pointeur sur une chaîne de caractères qui définit un préfixe,
- Le retour du message est l'index de la première chaîne trouvée respectant le préfixe ou un code d'erreur si aucune chaîne n'est trouvée.

```
lCode = SendMessage(hWnd, LB_GETCOUNT, 0, 0L)[10]
```

Ce message retournele nombre d'items présents dans la liste.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour du message est le nombre d'items présents dans la liste ou un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, LB_GETCURSEL, 0, 0L)
```

Ce message retourne l'index de l'item sélectionné. Ce message n'est pas appliquable si la liste est à sélection multiple.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour du message est l'index de l'item sélectionné dans la liste ou un code d'erreur si aucun item n'est sélectionné ou si la liste est à sélection multiple.

[10]Pour éviter toute confusion, un message préfixé par un GET fait une requête sur l'objet tandis qu'un message préfixé par un SET fait une mise à jour de l'objet.

```
lCode = SendMessage(hWnd, LB_GETITEMDATA, wIndex, 0L)
```

Ce message retourne un valeur su 32 bits propres à l'application associée à l'item d'index donné.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit l'index de l'item,
- Le quatrième n'est pas utilisé,
- Le retour du message est une valeur sur 32 bits associée à l'item ou un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, LB_GETITEMRECT, wIndex, (LONG)(LPRECT)lpRect)
```

Ce message retourne dans la structure pointée par "lpRect" les dimensions de l'item spécifié par l'index dans les coordonnées clients de l'objet.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit l'index de l'item,
- Le quatrième est un pointeur sur une structure de type "Rectangle",
- Le retour du message est un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, LB_GETSEL, wIndex, 0L)
```

Ce message retourne une valeur positive si l'item défini par l'index est sélectionné sinon zéro.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit l'index de l'item,
- Le quatrième n'est pas utilisé,
- Le retour du message est une valeur positive si l'item est sélectionné, zéro sinon ou un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, LB_GETSELCOUNT, 0, 0L)
```

Ce message retourne le nombre d'items sélectionnés. Il ne s'applique pas pour une liste à sélection unique

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour du message est le nombre d'items sélectionnés ou un code d'erreur si un

problème est rencontré ou si la liste est à sélection unique.

```
lCode = SendMessage(hWnd, LB_GETSELITEMS, wCount, (LONG)(LPINT)lpInt)
```

Ce message retourne les index des items sélectionnés dans le buffer pointé par "lpInt" dont la taille est définie par wParam.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est la taille du buffer dont l'unité est la taille d'un entier,
- Le quatrième est un pointeur sur un buffer d'entiers,
- Le retour du message est le nombre d'items placés dans le buffer ou un code d'erreur si un problème est rencontré ou si la liste est à sélection unique.

```
lCode = SendMessage(hWnd, LB_GETTEXT, wIndex, (LONG)(LPSTR)szString)
```

Ce message retourne la chaîne de l'item défini par l'index dans le buffer pointé par szBuffer. Ce dernier doit avoir une taille suffisante.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est l'index de l'item dont on veut copier la chaîne.
- Le quatrième est un pointeur sur une string.
- Le retour du message est la longeur de la chaîne en bytes excepté le caractère de fin de chaîne ou un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, LB_GETTEXTLEN, wIndex, 0L)
```

Ce message retourne la longueur de la chaîne de l'item défini par l'index.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est l'index de l'item pour lequel on veut connaître la longueur de la chaîne.
- Le quatrième n'est pas utilisé.
- Le retour du message est la longueur de la chaîne en bytes excepté le caractère de fin de chaîne ou un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, LB_GETTOPINDEX, 0, 0L)
```

Ce message retourne l'index du premier item visible dans la liste.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,

- Le quatrième n'est pas utilisé,
- Le retour du message est l'index du premier item visible.

```
lCode = SendMessage(hWnd, LB_HORIZONTALEXTENT, 0, 0L)
```

Ce message retourne le nombre de pixels dont l'objet interactif peut défiler horizontalement. Ce message est applicable si le style de l'objet contient WS_HSCROLL (défilement horizontal).

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour du message est le nombre de pixels.

```
SendMessage(hWnd, LB_INSERTSTRING, (WORD)nIndex, (LONG)(LPSTR)szString)
```

Ce message permet d'insérer une chaîne pointée par szString à la position définie par wIndex. Si wIndex est égal à -1, l'insertion se fait à la fin.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est l'index ou l'on veut insérer la chaîne,
- Le quatrième est un pointeur sur une chaîne,
- Le retour du message est l'index ou la chaîne a été insérée ou un code d'erreur si un problème est rencontré.

```
SendMessage(hWnd, LB_RESETCONTENT, 0, 0L)
```

Ce message supprime tous les items présents dans la liste.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé.

```
lCode = SendMessage(hWnd, LB_SELECTSTRING, (WORD)nIndex, (LONG)(LPSTR)szPrefix)
```

Ce message permet de changer la sélection courante par le premier item qui a le préfixe donné et qui se situe après l'index donné. Si l'index vaut -1, la recherche se fait à partir du début de la liste. Ce message n'est pas applicable à une liste à sélection multiple.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,

- Le troisième est l'index de départ,
- Le quatrième est un pointeur sur un préfixe,
- Le retour du message est l'index de l'item sélectionné ou un code d'erreur si un problème est rencontré ou si la liste est à sélection multiple.

```
lCode = SendMessage(hWnd, LB_SELITEMRANGE, wMode, lValue)
```

Ce message permet de sélectionner plusieurs items consécutifs. Ce message n'est pas applicable à une liste à sélection unique.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit si on sélectionne (différent de zéro) ou si on désélectionne (zéro),
- Le quatrième contient la valeur de l'index du premier item dans la partie basse et la valeur de l'index du dernier item dans la partie haute,
- Le retour du message est un code d'erreur si un problème est rencontré ou si la liste est à sélection unique.

```
SendMessage(hWnd, LB_SETCOLUMNWIDTH, wWidth, 0L)
```

Ce message définit la largeur des colonnes pour une liste à colonnes multiples.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit la largeur en pixels des colonnes,
- Le quatrième n'est pas utilisé.

```
lCode = SendMessage(hWnd, LB_SETCURSEL, (WORD)nIndex, 0L)
```

Ce message permet de sélectionner un nouvel item et le rend visible si la valeur de l'index est positive sinon il désélectionne l'item courant. Ce message n'est pas applicable à une liste à sélection multiple.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est l'index de l'item à sélectionner,
- Le quatrième n'est pas utilisé,
- Le retour du message est un code d'erreur si un problème est rencontré, ou si la liste est à sélection multiple.

```
SendMessage(hWnd, LB_SETHORIZONTALEXTENT, wWidth, 0L)
```

Ce message permet de définir la largeur de défilement horizontal de l'objet si celui-ci possède le style WS_HSCROLL.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit la largeur,
- Le quatrième n'est pas utilisé.

```
lCode = SendMessage(hWnd, LB_SETITEMDATA, wIndex, lValue)
```

Ce message définit la valeur spécifique associée à l'index donné.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est l'index de l'item,
- Le quatrième est la valeur définie sur 32 bits,
- Le retour du message est un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, LB_SETSEL, wMode, lValue)
```

Ce message permet de sélectionner (si wMode est différent de zéro) ou de désélectionner (si wMode égale zéro) l'item défini par l'index qui se trouve dans la partie basse de lValue ou tous les items si la valeur égale -1. Ce message n'est pas applicable à une liste à sélection unique.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit le mode,
- Le quatrième contient dans sa partie basse la valeur de l'index ou -1,
- Le retour du message est un code d'erreur si un problème est rencontré ou si la liste est à sélection unique.

```
lCode = SendMessage(hWnd, LB_SETTABSTOPS, wCount, (LONG)(LPINT)lpInt)
```

Ce message permet de définir la position des tabulations dans la liste. La position des tabulations est donnée par le pointeur lpInt et le nombre est défini par wCount.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième définit le nombre de tabulations,
- Le quatrième est un pointeur sur un tableau d'entiers ou chaque élément définit la position.
- Le retour du message est TRUE si toutes les tabulations sont positionnées sinon FALSE.

```
lCode = SendMessage(hWnd, LB_SETTOPINDEX, wIndex, 0L)
```

Ce message met l'item défini par l'index comme le premier élément visible de la liste.

- Le premier paramètre est l'objet interactif cible (fenêtre),
- Le deuxième correspond au type de message,
- Le troisième est l'index de l'item,
- Le quatrième n'est pas utilisé,
- Le retour du message est un code d'erreur si un problème est rencontré.

4. Messages de notification.

Etant donné que ces messages sont envoyés par l'objet interactif à l'application (fenêtre parent) et que cette dernière va éventuellement les traiter via la fonction associée à la fenêtre, le formalisme sera différent. Ainsi, les fonctions "SendMessage" ci-dessous seront envoyées par l'objet interactif.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LBN_DBLCLK, hWnd))
```

Ce message notifie la fenêtre parent que l'objet interactif a reçu un message de double-clic souris.

- Le premier paramètre est la fenêtre parent de l'OI,
- Le deuxième correspond au type de message,
- Le troisième est le numéro de l'objet interactif. Ce numéro doit être unique pour tous les OI fils de la fenêtre parent,
- Le quatrième est la combinaison du type de message de notification et de la référence sur l'OI (handle).

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LBN_ERRSPACE, hWnd))
```

Ce message notifie la fenêtre parent que l'objet interactif n'a pas su allouer suffisamment de mémoire pour stocker les items.

Pour les paramètres voir le précédent.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LBN_KILLFOCUS, hWnd))
```

Ce message notifie la fenêtre parent que l'objet interactif perd le focus.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LBN_SELCHANGE, hWnd))
```

Ce message notifie la fenêtre parent que la sélection a changé.

SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LBN_SETFOCUS, hWnd))

Ce message notifie la fenêtre parent que l'objet interactif a recu le focus.

5. Actions effectuées par l'utilisateur.

Sélection d'un élément dans la liste (sélection unique).

La sélection est effectuée au moyen d'un clic souris avec le bouton gauche sur un item ou au moyen du clavier avec les flèches vers le haut ou vers le bas.

Sélection de plusieurs éléments dans la liste (Sélection multiple)

Les sélections se font via la souris par un clic avec le bouton gauche sur les items ou via le clavier avec les flèches (haut et bas) et la barre d'espacement.

Défilement vertical

Si le nombre d'items est supérieur au nombre visualisés, une barre de défilement vertical apparaît à droite de la liste. Le défilement est effectué par un clic avec le bouton gauche sur les indicateurs (haut ou bas) ou dans les zones supérieure et inférieure à la zone centrale, en déplaçant la zone centrale via le bouton gauche enfoncé, ou avec le clavier via les touches haut, bas, PgUp, PgDn, Home et End.

Ainsi, pour spécifier nos différents objets interactifs nous procéderons de la même façon, c'est à dire, une définition générale de l'objet, une définition des messages que l'objet peut supporter et les actions utilisateurs. En fait, si une définition générale peut être donnée, on verra qu'il est plus difficile de spécifier de manière précise tous les messages que l'objet devra supporter.

b) Spécification de l'objet interactif de type "Meter".

L'objet interactif de type "Meter" a pour but d'informer l'utilisateur de l'état d'avancement d'une tâche telle que: copie de fichiers, traitement mathématique ... . Ce type d'objet se présente sous la forme d'une réglette où l'état d'avancement est indiqué par une valeur représentant le pourcentage du travail effectué et par un changement du "look" de l'objet comme indiqué ci-dessous.
Cet objet sera de type statique dans la mesure où l'utilisateur n'a pas d'interaction possible sur lui.

1. Représentation graphique.

2. Attributs de spécialisation de l'objet.

Ce type d'objet n'a pas de style particulier.

3. Messages de contrôle.

```
SendMessage(hWnd, MM_GETCURRENTVALUE, 0, 0L)
```

Ce message permet de retourner la valeur courante.

- Le premier paramètre correspond à l'objet interactif,
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour du message correspond à la valeur courante.

```
SendMessage(hWnd, MM_GETMAXVALUE, 0, 0L)
```

Ce message permet de retourner la valeur maximale.

- Le premier paramètre correspond à l'objet interactif,
- Le deuxième correspond au type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour du message correspond à la valeur maximale.

```
SendMessage(hWnd, MM_SETCURRENTVALUE, wCurrentValue, 0L)
```

Ce message permet de définir la valeur courante.

- Le premier paramètre correspond à l'objet interactif,
- Le deuxième correspond au type de message,
- Le troisième correspond à la valeur courante,
- Le quatrième n'est pas utilisé.

```
SendMessage(hWnd, MM_SETMAXVALUE, wMaxValue, 0L)
```

Ce message permet de définir la valeur maximale.

- Le premier paramètre correspond à l'objet interactif,
- Le deuxième correspond au type de message,
- Le troisième correspond à la valeur de la borne maximale,
- Le quatrième n'est pas utilisé.

4. Message de notification.

Cet objet interactif ne possède pas de messages de notification.

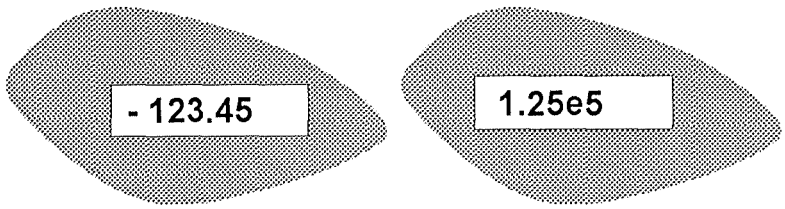
5. Actions effectuées par l'utilisateur.

Aucune action utilisateur.

c) Spécification de l'objet interactif de type "EditFloat".

Avant de spécifier ce type d'objet interactif, nous allons préciser les objets interactifs existant dans l'environnement *Windows* qui supportent l'acquisition de valeurs définies dans l'ensemble des réels. Comme support à ce type d'acquisition nous avons les objets interactifs de type Edit, Combo et dans une moindre mesure le type List. Cependant, ces OIs ne sont pas bien adaptés car ils ne font pas de vérification sur les caractères entrés, ne supportent que des variables de type "String" pour être initialisés et ne fournissent à l'application que des valeurs de type "String" également.
Ainsi l'OI "EditFloat" permettra une vérification sur les caractères entrés (fonction du type) et sur le respect des bornes, une initialisation par des valeurs de type réel, un retour vers l'application par une valeur de type réel et un bornage inférieur et supérieur. Nous limiterons le domaine de cet OI au niveau des valeurs acceptables pour le type "Float". Nous estimons en effet que des valeurs de type "Double" sont rarement acquises au niveau de l'interface hômme-machine.

1. Représentation graphique.



2. Attributs de spécialisation de l'objet.

FS_BEEP	Emet un son lors de la frappe d'un caractère erroné.
FS_STRICT	Limite l'entrée de caractère suivant la définition des bornes.

Les styles suivants sont hérités de la classe Edit :

ES_PASSWORD	Affiche une étoile pour chaque caractère entré.
ES_AUTOHSCROLL	Le texte défile automatiquement lorsque le dernier caractère entré arrive à la fin de la zone visible.
ES_NOHIDESEL	Normalement, lors de la perte du focus, le texte sélectionné reprend l'aspect normal, et lors de la prise de focus, il inverse le texte sélectionné. Cette option supprime cette action.

ES_OEMCONVERT	Le texte entré est converti de l'ANSI en OEM mode.

Les styles suivants sont hérités de la classe Window :

WS_BORDER	Affiche un bord à l'objet.
WS_TABSTOP	Permet la prise (ou la perte) de focus via la touche TAB.
WS_GROUP	Permet de définir un groupe à l'intérieur duquel on peut se déplacer avec les flèches.

3. messages de contrôle.

```
lCode = SendMessage(hWnd, FM_CHECKCURRENTVALUE, 0, 0L)
```

Ce message permet de tester la validité de la valeur entrée. La validité porte sur le type et sur le respect des bornes.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour de la fonction est un code qui donne l'état de la valeur courante,
  F_OK si la valeur est correcte,
  F_NULL si aucune valeur n'est entrée,
  F_BADFORMAT si la valeur n'est pas un float,
  F_OUTOFRANGE si la valeur est hors limite,
  F_ERR si une erreur est survenue.

```
lCode = SendMessage(hWnd, FM_GETCURRENTVALUE, sizeof(float), LONG)(LPFLOAT)lpFloat)
```

Ce message permet de retourner la valeur courante dans le buffer pointé par lpFloat.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit la taille du buffer,
- Le quatrième est un pointeur sur un float,
- Le retour de la fonction est un code qui donne l'état de la valeur courante.
  (Cfr ci-dessus)

```
lCode = SendMessage(hWnd, FM_GETMAXVALUE, sizeof(float), (LONG)(LPFLOAT)lpFloat)
```

Ce message permet de retourner la borne supérieure dans le buffer pointé par lpFloat.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit la taille du buffer,
- Le quatrième est un pointeur sur un float.

```
lCode = SendMessage(hWnd, FM_GETMINVALUE, sizeof(float), (LONG)(LPFLOAT)lpFloat)
```

Ce message permet de retourner la borne inférieure dans le buffer pointé par lpFloat.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit la taille du buffer,
- Le quatrième est un pointeur sur un float.

```
lCode = SendMessage(hWnd, FM_SETCURRENTVALUE, 0, (LONG)(LPFLOAT)lpFloat)
```

Ce message permet d'initialiser la valeur courante par la valeur pointée par lpFloat. Si lpFloat est NULL alors la valeur est initialisée à NULL.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur un float,
- Le retour de la fonction est F_OK si la valeur est correcte sinon le code est F_OUTOFRANGE.

```
lCode = SendMessage(hWnd, FM_SETMAXVALUE, 0, (LONG)(LPFLOAT)lpFloat)
```

Ce message permet d'initialiser la borne supérieure par la valeur pointée par lpFloat. Si lpFloat est NULL alors la valeur est initialisée au maximum .

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur un float,
- Le retour de la fonction est la borne supérieure.

```
lCode = SendMessage(hWnd, FM_SETMINVALUE, 0, (LONG)(LPFLOAT)lpFloat)
```

Ce message permet d'initialiser la borne inférieure par la valeur pointée par lpFloat. Si lpFloat est NULL alors la valeur est initialisée au minimum.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,

- Le quatrième est un pointeur sur un float,
- Le retour de la fonction est la borne inférieure.

Afin d'éviter de devoir définir des variables pour pouvoir passer les valeurs par adresse, une fonction sera définie afin de pouvoir initialiser les valeurs via une constante. Le prototype de cette fonction est donné ci-dessous :

```
LONG FAR PASCAL FloatAdress(FLOAT);
```

Avec comme exemple d'utilisation:

```
...
SendMessage(hWnd, FM_SETMINVALUE, 0, (*floatAdress)(123.56));
...
```

4. Messages de notification.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(FN_BADCHAR, hWnd))
```

Ce message notifie la fenêtre parent de la frappe d'un caractère erroné.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(FN_BADFORMAT, hWnd))
```

Ce message notifie la fenêtre parent que la valeur courante n'est pas un float.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(FN_BADPOSCHAR, hWnd))
```

Ce message notifie la fenêtre parent de la frappe d'un caractère dans une position erronée.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(FN_OUTOFRANGE, hWnd))
```

Ce message notifie la fenêtre parent que la valeur courante est en-dehors des bornes fixées.

De plus certains messages qui sont définis pour l'OI "Edit" seront également supportés.

5. Actions effectuées par l'utilisateur.

Les actions utilisateur de cet objet sont les mêmes que pour l'objet dont il hérite (Edit).

Frappe d'un caractère

Si la frappe correspond à un caractère imprimable qui respecte les contraintes du type, le texte est mis à jour. Si la frappe correspond à un caractère de contrôle (déplacement ou sélection) l'action est effectuée.

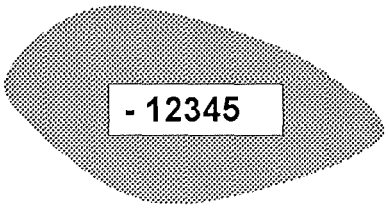
Sélection partielle ou complète du texte entré

La sélection est effectuée par lun clic souris sur le début de la sélection et en se déplaçant avec le bouton gauche enfoncé ou par un double-clic sur un mot ce qui le sélectionne. Elle peut être réalisée par le clavier en maintenant la touche "SHIFT" enfoncée lors du déplacement. Cette technique est utile pour la suppression du texte et pour le "Copier/Collé".

d) Spécification de l'objet interactif de type "EditLong".

Ce type d'OI reprend en partie les spécifications de l'OI de type "EditFloat". Celui-ci sera limité à l'acquisition de valeurs entières positives ou négatives suivant les valeurs des bornes. Par défaut les bornes sont initialisées par les valeurs admissibles c'est-à-dire -2^{31} à $2^{31}-1$. Notons également que, pour des raisons de standardisation, les valeurs passées à l'objet le seront via leurs adresses.

1. Représentation graphique.



2. Attributs de spécialisation de l'objet.

LGS_BEEP	Emet un son lors de la frappe d'un caractère erroné.
LGS_STRICT	Limite l'entrée de caractère suivant la définition des bornes.

Les styles suivants sont hérités de la classe Edit :

(Cfr la classe EditFloat)

Les styles suivants sont hérités de la classe Window :

(Cfr la classe EditFloat)

3. Messages de contrôle.

```
lCode = SendMessage(hWnd, LGM_CHECKCURRENTVALUE, 0, 0L)
```

Ce message permet de tester la validité de la valeur entrée. La validité porte sur le type et sur le respect des bornes.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour de la fonction est un code qui donne l'état de la valeur courante,

LG_OK si la valeur est correcte,
LG_NULL si aucune valeur n'est entrée,
LG_BADFORMAT si la valeur n'est pas un long,
LG_OUTOFRANGE si la valeur est hors limite,
LG_ERR si une erreur est survenue.

```
lCode = SendMessage(hWnd, LGM_GETCURRENTVALUE, sizeof(LONG),(LONG)(LPLONG)lpLong)
```

Ce message permet de retourner la valeur courante dans le buffer pointé par lpLong.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième défini la taille du buffer,
- Le quatrième est un pointeur sur un long,
- Le retour de la fonction est un code qui donne l'état de la valeur courante. (Cfr ci-dessus)

```
lCode = SendMessage(hWnd, LGM_GETMAXVALUE, sizeof(LONG), (LONG)(LPLONG)lpLong)
```

Ce message permet de retourner la borne supérieure dans le buffer pointé par lpLong.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit la taille du buffer,
- Le quatrième est un pointeur sur un long.

```
lCode = SendMessage(hWnd, LGM_GETMINVALUE, sizeof(LONG), (LONG)(LPLONG)lpLong)
```

Ce message permet de retourner la borne inférieure dans le buffer pointé par lpLong.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième défini la taille du buffer,
- Le quatrième est un pointeur sur un long.

```
lCode = SendMessage(hWnd, LGM_SETCURRENTVALUE, 0, (LONG)(LPFLOAT)lpLong)
```

Ce message permet d'initialiser la valeur courante par la valeur pointée par lpLong. Si lpLong est NULL alors la valeur est initialisée à NULL.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur un long,
- Le retour de la fonction est LG_OK si la valeur est correcte sinon le code est

LG_OUTOFRANGE.

```
lCode = SendMessage(hWnd, LGM_SETMAXVALUE, 0, (LONG)(LPLONG)lpLong)
```

Ce message permet d'initialiser la borne supérieure par la valeur pointée par lpLong. Si lpLong est NULL alors la valeur est initialisée au maximum .

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur un long,
- Le retour de la fonction est la borne supérieure.

```
lCode = SendMessage(hWnd, LGM_SETMINVALUE, 0, (LONG)(LPLONG)lpLong)
```

Ce message permet d'initialiser la borne inférieure par la valeur pointée par lpLong. Si lpLong est NULL alors la valeur est initialisée au minimum.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur un long,
- Le retour de la fonction est la borne inférieure.

Afin d'éviter de devoir définir des variables pour pouvoir passer les valeurs par adresse, une fonction sera définie afin de pouvoir initialiser les valeurs via une constante. Le prototype de cette fonction est donnée si dessous:

```
LONG FAR PASCAL LongAdress(LONG);
```

Avec comme exemple d'utilisation:

```
...
SendMessage(hWnd, LGM_SETMINVALUE, 0, (*longAdress)(12356));
...
```

4. Messages de notification.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LGN_BADCHAR, hWnd))
```

Ce message notifie la fenêtre parent de la frappe d'un caractère erroné.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LGN_BADFORMAT, hWnd))
```

Ce message notifie la fenêtre parent que la valeur courante n'est pas un long.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LGN_BADPOSCHAR, hWnd))
```

Ce message notifie la fenêtre parent de la frappe d'un caractère dans une position erronée.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(LGN_OUTOFRANGE, hWnd))
```

Ce message notifie la fenêtre parent que la valeur courante est en-dehors des bornes fixées.

De plus certains messages qui sont définis pour l'OI "Edit" seront également supportés.

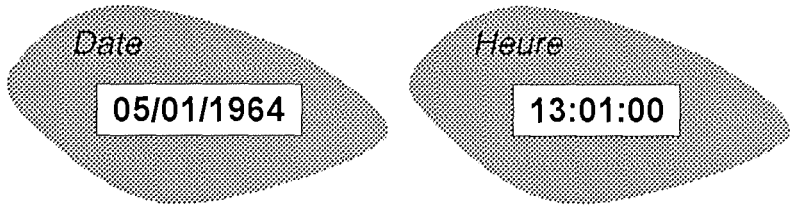
5. Actions effectuées par l'utilisateur.

(Voir l'objet "EditFloat")

e) Spécification de l'objet interactif de type "DateTime".

Ce type d'OI permettra la saisie de la date ou de l'heure suivant le format spécifié dans l'environnement *Windows* (DMY MDY YMD). A l'affichage, un masque de saisie (séparateur défini suivant l'environnement) sera présenté comme sur le graphique ci-dessous. Une vérification lexicale et sémantique sera effectuée sur la valeur entrée. Dans l'environnement Windows, il existe deux modes de représentation du format de la date: court et long[11]. Ici seul le format de type court sera supporté. De même pour l'heure, seul le mode Européen sera utilisé[12]. De ce fait, deux structures de données seront définies.
Lors du placement de l'OI via l'éditeur de boîte de dialogue, on pourra contraindre les dimensions de l'objet. En effet, celui-ci possède un format bien défini quant à sa longueur.

1. Représentation graphique.



2. Attributs de spécialisation de l'objet.

DTS_ADJUST	Ajuste la longueur de la fenêtre en fonction du format.
DTS_BEEP	Emet un son lors de la frappe d'un caractère erroné.
DTS_DATE | DTS_TIME	Définit le type de valeur.
DTS_STRICT	Limite l'entrée de caractère suivant la définition des bornes.

Les styles suivants sont hérités de la classe Edit :

(Cfr la classe EditFloat)

Les styles suivants sont hérités de la classe Window :

(Cfr la classe EditFloat)

---

[11]Exemple du format date de type court: "05/01/1964"; de type long: "Monday, 05 january, 1964".
[12]Exemple du format de type Européen: "16:53:55"; de type Américain: "04:53:55 PM".

3. Messages de contrôle.

lCode = SendMessage(hWnd, DTM_CHECKCURRENTVALUE, 0, 0L)

Ce message permet de tester la validité de la valeur entrée. La validité porte sur le type et sur le respect des bornes.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour de la fonction est un code qui donne l'état de la valeur courante,
  DT_OK si la valeur est correcte,
  DT_NULL si aucune valeur n'est entrée,
  DT_BADFORMAT si la valeur n'est pas un long,
  DT_OUTOFRANGE si la valeur est hors limite,
  DT_ERR si une erreur est survenue.

lCode = SendMessage(hWnd, DTM_GETCURRENTVALUE, sizeof(DATE[13]), (LONG)(LPDATE)lpDate

Ce message permet de retourner la valeur courante dans le buffer pointé par lpDate.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit la taille du buffer,
- Le quatrième est un pointeur sur une structure de type date,
- Le retour de la fonction est un code qui donne l'état de la valeur courante.
  (Cfr ci-dessus)

lCode = SendMessage(hWnd, DTM_GETMAXVALUE, sizeof(DATE), (LONG)(LPDATE)lpDate)

Ce message permet de retourner la borne supérieure dans le buffer pointé par lpDate.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit la taille du buffer,
- Le quatrième est un pointeur sur une structure de type date.

lCode = SendMessage(hWnd, DTM_GETMINVALUE, sizeof(DATE), (LONG)(LPDATE)lpDate)

Ce message permet de retourner la borne inférieure dans le buffer pointé par lpDate.

- Le premier paramètre est l'objet interactif,

[13] DATE ou TIME suivant le type.

- Le deuxième est le type de message,
- Le troisième définit la taille du buffer
- Le quatrième est un pointeur sur une structure de type date.

```
lCode = SendMessage(hWnd, DTM_SETCURRENTNOW, 0, 0L)
```

Ce message permet d'initialiser la valeur courante par la date ou l'heure actuelle.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé.

```
lCode = SendMessage(hWnd, DTM_SETCURRENTVALUE, 0, (LONG)(LPDATE)lpDate)
```

Ce message permet d'initialiser la valeur courante par la valeur pointée par lpDate. Si lpDate est NULL alors la valeur est initialisée à NULL.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur une structure de type date,
- Le retour de la fonction est LG_OK si la valeur est correcte sinon le code est LG_OUTOFRANGE.

```
lCode = SendMessage(hWnd, DTM_SETMAXVALUE, 0, (LONG)(LPDATE)lpDate)
```

Ce message permet d'initialiser la borne supérieure par la valeur pointée par lpDate. Si lpDate est NULL alors la valeur n'est pas limitée.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur une structure de type date,
- Le retour de la fonction est la borne supérieure.

```
lCode = SendMessage(hWnd, DTM_SETMINVALUE, 0, (LONG)(LPDATE)lpDate)
```

Ce message permet d'initialiser la borne inférieure par la valeur pointée par lpDate. Si lpDate est NULL alors la valeur n'est pas limitée.

- Le premier paramètre est l'objet interactif.
- Le deuxième est le type de message.
- Le troisième n'est pas utilisé.
- Le quatrième est un pointeur sur une structure de type date.

- Le retour de la fonction est la borne inférieure.

Afin d'éviter de devoir définir des variables pour pouvoir passer les valeurs par adresse, deux fonctions seront définies dans le but d'initialiser la valeur via une constante. Le prototype de ces fonctions est donné ci-dessous :

```
LONG FAR PASCAL DateAdress(int, int, int);
LONG FAR PASCAL TimeAdress(int, int, int);
```

Avec comme exemple d'utilisation:

```
...
SendMessage(hWnd, DTM_SETMINVALUE, 0, (*dateAdress)(5, 1, 1964));
...
```

4. Messages de notification.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(DTN_BADCHAR, hWnd))
```

Ce message notifie la fenêtre parent de la frappe d'un caractère erroné.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(DTN_BADFORMAT, hWnd))
```

Ce message notifie la fenêtre parent que la valeur courante n'est pas une date.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(DTN_OUTOFRANGE, hWnd))
```

Ce message notifie la fenêtre parent que la valeur courante est en-dehors des bornes fixées.

De plus certains messages qui sont définis pour l'OI "Edit" seront également supportés.

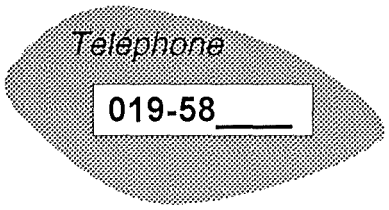
5. Actions effectuées par l'utilisateur.

(Voir l'objet "EditFloat")

f) Spécification de l'objet interactif de type "Mask".

Ce type d'OI trouve son intérêt lors de l'acquisition d'informations qui doivent respecter un certain format. Ce type d'OI est souvent présent dans des librairies de gestion d'interface de type "texte". Le masque de saisie sera composé d'un caractère générique ('_') et de caractères prédéfinis. Une contrainte pourra être appliquée sur le caractère générique (Numérique, Numérique + Espace, Alpha-numérique, Alpha). De plus, une vérification sémantique de la variable pourra être effectuée via une "Callback function".
Notons également, que le OI de type Date&Time peut être vu comme un sous-type de celui-ci.

1. Représentation graphique.



2. Attributs de spécialisation de l'objet.

MKS_ADJUST	Ajuste la longueur de la fenêtre en fonction du format.
MKS_BEEP	Emet un son lors de la frappe d'un caractère erroné.

Les styles suivants sont hérités de la classe Edit :

(Cfr la classe EditFloat)

Les styles suivants sont hérités de la classe Window :

(Cfr la classe EditFloat)

3. Messages de contrôle.

```
lCode = SendMessage(hWnd, MKM_CHECKCURRENTVALUE, 0, 0L)
```

Ce message retourne l'état de la variable.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour de la fonction est un code.

```
lCode = SendMessage(hWnd, MKM_GETMASK, wSize, (LONG)(LPSTR)szMask)
```

Ce message permet de retourner le masque de saisie dans le buffer pointé par szMask.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit la taille du buffer,
- Le quatrième pointe sur un buffer,
- Le retour de la fonction est le nombre de bytes copiés ou un code d'erreur si un problème est rencontré.

```
lCode = SendMessage(hWnd, MKM_GETMASKLENGTH, 0, 0L)
```

Ce message permet de retourner la longueur du masque en nombre de caractères.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour de la fonction est la taille du masque.

```
SendMessage(hWnd, MKM_SETCALLBACK, 0, (LONG)(LPPROC)lpProc)
```

Ce message fourni le pointeur sur la fonction de validation. Si lpProc est NULL il n'y a pas ou plus de fonction de validation.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième est un pointeur sur la fonction de validation.

```
lCode = SendMessage(hWnd, MKM_SETMASK, 0, (LONG)(LPSTR)szMask)
```

Ce message permet de définir le masque de saisie.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième pointe sur une chaîne de caractères qui définit le masque de saisie.
- Le retour de la fonction est le code MK_ERR si un problème est rencontré sinon le code sera MK_OK. Au niveau des contraintes sur les caractères nous avons défini les symboles suivant:

    - Pour les valeurs numériques: #
    - Pour les valeurs alphabétiques: @
    - Pour les valeurs numériques + espace: |
    - Pour les valeurs alpha-numériques: &

  Notons qu'il serait peut être utile de pouvoir paramétrer ces symboles afin de garantir tout caractère possible au niveau du masque.

Pour initialiser ou retourner la valeur courante les messages WM_SETTEXT et WM_GETTEXT seront utilisés.

4. Messages de notification.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(MKN_BADCHAR, hWnd))
```

Ce message notifie la fenêtre parent de la frappe d'un caractère erroné.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(MKN_BADFORMAT, hWnd))
```

Ce message notifie la fenêtre parent que la valeur courante ne respecte pas les contraintes définies par le masque.

De plus certains messages qui sont définis pour l'OI "Edit" seront également supportés.

Pour être complet, il nous reste à spécifier le format de la fonction de validation.

```
LONG FAR PASCAL CallBackValidation(hWnd, LONG)
```

- Le premier paramètre est la référence à l'objet (HANDLE).
- Le second est de type LONG et est en fait un pointeur sur la chaîne de caractères.
- Le retour est un code qui définit l'état de la variable.

5. Actions effectuées par l'utilisateur.

(Voir l'objet "EditFloat")

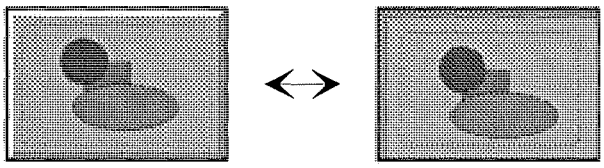
g) Spécification de l'objet interactif de type "GraphicButton".

Au niveau des OIs standards dans l'environnement Windows nous trouvons l'OI de type "Button" qui peut être spécialisé suivant trois modes : le mode "Push Button", le mode "Radio Button" et enfin le mode "Check Button". Ce type d'OI est caractérisé au niveau de sa présentation par une chaîne de caractères. Si on veut qu'il soit visualisé sous forme d'un graphique ou d'une icône il est nécessaire de gérer cette forme de présentation en-dehors de l'OI (dans l'application en spécifiant le style Owner-draw). Dès lors, il semble intéressant de développer un OI qui puisse gérer sa présentation sous forme graphique. Cependant, par rapport à la version de l'OI "Button" qui voit sa présentation changer en fonction du mode, celui qu'on se propose de définir ne changera pas de présentation en fonction de son mode. Seul, son comportement sera différent en fonction du mode choisi (voir graphique).
Du fait que cet OI est purement graphique, il devient difficile de lui associer un code clavier.

Au niveau des messages que cet OI devra supporter nous trouverons les messages définis pour le bouton standard que nous reprenons ci-après ainsi que certains messages qui lui sont spécifiques.

1. Représentation graphique.

Cette représentation est valable pour les trois modes.



Relaché Enfoncé

2. Attributs de spécialisation de l'objet.

GBS_AUTOCHECKBOX	Définit le button comme une boîte à cocher automatique.
GBS_AUTORADIOBUTTON	Définit le button comme un bouton radio. Les autres radio boutons du groupe sont désélectionnés.
GBS_CHECKBOX	Définit le button comme une boîte à cocher. Celle-ci sera éventuellement cochée par l'application.

GBS_CLIP	Détermine la taille du bouton. Dans ce mode, la taille est définie par l'utilisateur et l'image est éventuellement "coupée".
GBS_DEFPUSHBUTTON	Définit le button comme étant de type push button par défaut.
GBS_PUSHBUTTON	Définit le button comme étant de type push button.
GBS_RADIOBUTTON	Définit le button comme un bouton radio. Celui-ci sera éventuellement coché par l'application.
GBS_RESIZE	La taille de la fenêtre est ajustée en fonction de l'image.
GBS_STRETCH	La taille de la fenêtre est définie par l'utilisateur. L'image est déformée en conséquence.

Les styles suivants sont hérités de la classe Window :

WS_GROUP, WS_TABSTOP, WS_BORDER, WS_DISABLED.

3. Messages de contrôle.

```
lCode = SendMessage(hWnd, GBM_GETCHECK, 0, 0L)
```

Ce message retourne l'état du bouton de type radio ou check.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,
- Le retour de la fonction est une valeur différente de zéro si le bouton est marqué sinon zéro.

```
SendMessage(hWnd, GBM_SETCHECK, wCheck, 0L)
```

Ce message permet de marquer le bouton de type radio ou check si la valeur de wCheck est différente de zéro sinon le bouton est désélectionné.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit si le bouton doit être sélectionné ou pas,
- Le quatrième n'est pas utilisé.

```
SendMessage(hWnd, GBM_SETSTYLE, wStyle, lRedraw)
```

Ce message permet de changer le style d'un bouton.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit le style du bouton,
- Le quatrième détermine si le bouton doit être redessiné si lRedraw est différent de zéro sinon il n'est pas redessiné.

Pour définir l'image du bouton on envoie le message WM_SETTEXT avec comme argument le nom du fichier de type bitmap sans extension.

4. Messages de notification.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(GBN_CLICKED, hWnd))
```

Ce message notifie la fenêtre parent que le bouton a été sélectionné par un clic souris.

```
SendMessage(hWndParent, WM_COMMAND, ID,MAKELONG(GBN_DOUBLECLICKED, hWnd))
```

Ce message notifie la fenêtre parent que le bouton a été sélectionné par un double-clic souris.

5. Actions effectuées par l'utilisateur.

```
Sélection du bouton
```

Cette sélection est effectuée par un clic ou un double clic souris ou par le clavier grâce à la barre d'espacement si le bouton possède le focus.

h) Spécification de l'objet interactif de type "Curve2D".

Nous avons séparé les graphiques 2D des graphiques 3D pour plusieurs raisons. Tout d'abord, le type de données manipulé est différent de par sa structure. Ensuite, les algorithmes mis en oeuvre diffèrent. Enfin, il sera ainsi possible de limiter la taille des objets interactifs.
Ce type d'objet interactif permettra d'afficher des courbes, des iso-courbes avec la possibilité d'effectuer un lissage si cela est nécessaire. Les valeurs passées seront des données provenant directement de l'application afin d'éviter de les dupliquer et pour éviter de limiter les OIs à des types de données pré-définis . Cela nous obligera à utiliser des "Callback functions". On aura la possibilité de définir l'espace de représentation en vue globale ou de spécifier les limites. On aura la possibilité de choisir le système d'axes; de définir les attributs d'affichages à savoir: le type de légende, le type d'échelles, les intitulés des axes; de placer des commentaires; de choisir les polices, formats et couleurs d'affichage. Les options qui sont facultatives seront définies par défaut.
L'objet aura la possibilité d'être un objet interactif, c'est-à-dire, de retourner certaines informations en fonction de la zone sélectionnée. Dans un deuxième temps, on donnera à l'utilisateur la possibilité de modifier la plupart des paramètres d'affichage.
Etant donné les possibilités que cet objet doit supporter, il nous semble raisonnable d'affirmer qu'il serait préférable de voir cet objet comme une coopération de plusieurs objets interactifs. Ainsi nous avons sélectionné 4 types d'objets interactifs qui sont:

- Frame: cet objet va intégrer les autres objets. Cette intégration est visuelle d'une part, et c'est avec lui que, d'autre part, l'application va dialoguer d'autre part.

- Scale: cet objet permet de définir et visualiser les échelles en abscisse et en ordonnée du graphique.

- Graph2D: cet objet va être responsable de l'affichage des courbes.

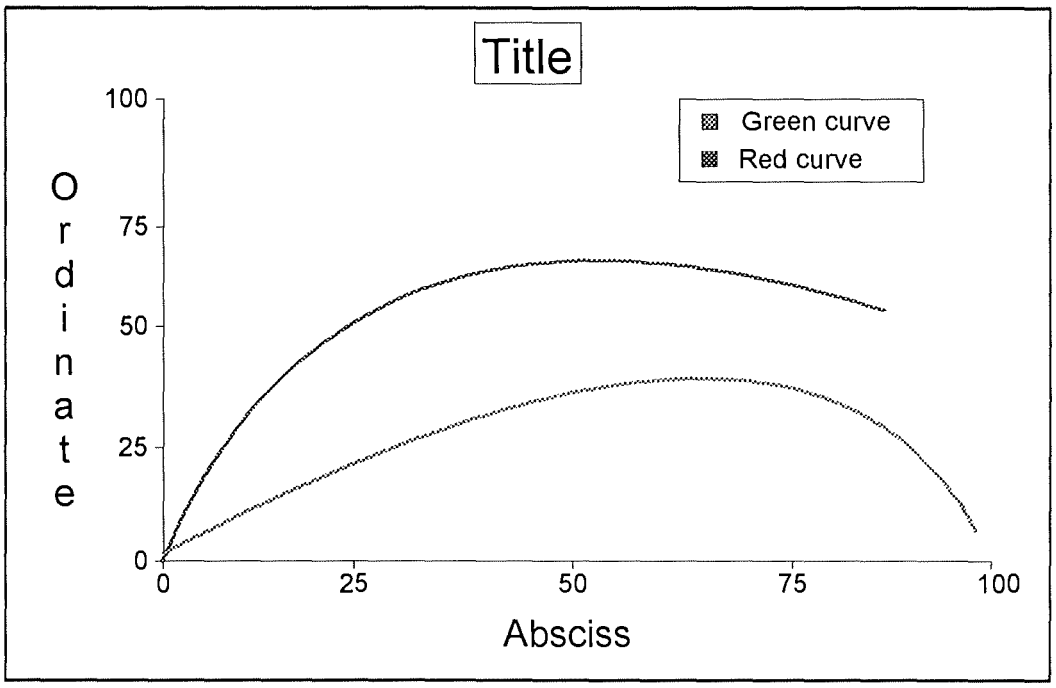
- Legend: cet objet va afficher les intitulés des courbes.

On remarque immédiatement que certains objets définis pour ce type de graphique pourront être réutilisés pour la visualisation de graphiques tri-dimensionnels.

Notons enfin que la spécification pourra évoluer en fonction des problèmes rencontrés.

Aggrégat Frame

1. Représentation graphique (pour l'ensemble des objets).



2. Attributs de spécialisation de l'objet de type FRAME.

FRS_ANISOTROPIC	Pas de contrainte sur les échelles.
FRS_BORDERTITLE	Le titre est encadré.
FRS_BOTTOMORDONNED	L'échelle des ordonnées se situe en bas.
FRS_BOTTOMTITLE	Le titre est situé en bas.
FRS_ISOTROPIC	Les échelles sont respectées.
FRS_LEFTABSCISS	L'échelle des abscisses est à gauche.
FRS_NOTITLE	Il n'y a pas de titre.
FRS_RIGHTABSCISS	L'échelle des abscisses est à droite.
FRS_TOPORDINATE	L'échelle des ordonnées se situe en haut.
FRS_TOPTITLE	Le titre est situé en haut.

3. Messages de contrôle de l'objet de type FRAME.

Pas de messages (on définira les messages associés à chaque objet)

4. Messages de notification de l'objet de type FRAME.

Pas de messages.

Aggrégat Scale

1. Représentation graphique.

(Voir Frame)

2. Attributs de spécialisation de l'objet de type SCALE

SCS_ABOVE	L'intitulé de l'échelle sera placé au dessus ou en dessous suivant la position de l'échelle.
SCS_ABSCISS	L'échelle est définie comme abscisse.
SCS_ASIDE	L'intitulé de l'échelle sera placé à coté.
SCS_APPROCHED	Les bornes seront approchées.
SCS_HORI	L'intitulé est placé horizontalement.
SCS_INCLB	L'accroissement se fait vers la gauche ou vers le bas suivant le type d'échelle.
SCS_INCRT	L'accroissement se fait vers la droite ou vers le haut suivant le type de l'échelle.
SCS_LINEAR	L'échelle est de type linéaire.
SCS_LOGA	L'échelle est de type logarithmique.
SCS_NONE	Il n'y a pas d'intitulé.
SCS_ORDINATE	L'échelle est définie comme ordonnée.
SCS_REAL	Les bornes ne sont pas arrondies.

SCS_SYMETRIC	Les valeurs des bornes inférieure et supérieure sont égales au signe près.
SCS_VERT	L'intitulé est placé verticalement.

3. Messages de contrôle.

```
SendMessage(hWnd, SCS_GETMAX, sizeof(float), (LONG)(LPFLOAT)lpFloat)
```

Ce message retourne la valeur de la borne supérieure dans le buffer pointé par lpFloat.

```
SendMessage(hWnd, SCS_GETMIN, sizeof(float), (LONG)(LPFLOAT)lpFloat)
```

Ce message retourne la valeur de la borne inférieure dans le buffer pointé par lpFloat.

```
SendMessage(hWnd, SCS_SETMAX, 0, (LONG)(LPFLOAT)lpFloat)
```

Ce message définit la valeur de la borne supérieure par la valeur pointée par lpFloat. Si lpFloat est NULL, alors la borne est définie par le maximum.

```
SendMessage(hWnd, SCS_SETMIN, 0, (LONG)(LPFLOAT)lpFloat)
```

Ce message définit la valeur de la borne inférieure par la valeur pointée par lpFloat. Si lpFloat est NULL, alors la borne est définie par le minimum.

```
SendMessage(hWnd, SCS_SETGLOBAL, 0, 0)
```

Ce message définit les bornes pour avoir la vue globale.

4. Messages de notification.

Pas de messages.

Aggrégat Legend

1. Représentation graphique.

(Voir Frame)

2. Attributs de spécialisation de l'objet de type LEGEND.

LD_BOTTOM	La légende est positionnée en bas.

LD_INSIDE	La légende est positionnée dans le graphique.
LD_LEFT	La légende est positionnée à gauche.
LD_TOP	La légende est positionnée en haut.
LD_RIGHT	La légende est positionnée à droite.

3. Messages de contrôle.

Pas de messages.

4. Messages de notification.

Les messages définis ci-dessous sont particuliers. Ils sont du même type que le message WM_CTLCOLOR qui est envoyé à la fenêtre parent d'un objet afin que sa présentation en soit modifiée.

```
SendMessage(hWndParent, LDN_SETCOLORCURVE, wCount, (LONG)(LPLONG)lpColort)
```

Ce message est envoyé à la fenêtre parent pour définir la couleur de la courbe de numéro wCount.

```
SendMessage(hWndParent, LDN_SETTEXTCURVE, wCount, (LONG)(LPSTR)szText)
```

Ce message est envoyé à la fenêtre parent pour définir le texte de la courbe de numéro wCount.

Aggrégat Curve2D

1. Représentation graphique.

(Voir Frame)

2. Attributs de spécialisation de l'objet de type CURVE2D.

C2D_CURVE	Le tracé est en mode courbe (lissage).
C2D_GRID	Une grille est affichée. La maille est définie par les échelles.
C2D_ISOCURVE	Le tracé est en mode iso-courbe.

C2D_LINE	Le tracé est en mode ligne.
C2D_POINT	Le tracé est en mode point.
C2D_ZOOM	Lors du déplacement de la souris avec le bouton gauche enfoncé, un rectangle se dessine.

3. Messages de contrôle.

```
SendMessage(hWnd, C2DM_GETPOS, sizeof(FPOINT), (LONG)(LPFPOINT)lpFpoint)
```

Ce message retourne la position du curseur dans le buffer de type pré-défini.

```
SendMessage(hWnd, C2DM_REDRAW, wCount, 0L)
```

Ce message force l'objet à redessiner la courbe de numéro wCount. Si wCount est négatif alors toutes les courbes sont retracées.

```
SendMessage(hWnd, C2DM_SETCALLBACK, wCount, (LONG)(LPPROC)lpProc)
```

Ce message fournit le pointeur de la call back function pour la courbe de numéro wCount. Si lpProc est NULL, alors la courbe est supprimée.

4. Messages de notification.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(C2DN_DOUBLECLICKED, hWnd))
```

Ce message notifie la fenêtre parent que l'objet à reçu un double-clic souris.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(C2DN_CLICKED, hWnd))
```

Ce message notifie la fenêtre parent que le bouton à reçu un clic souris.

Il nous reste à définir le format de la call back function:

```
LONG FAR PASCAL CallBackInformation(HWND, WORD, LONG, LONG)
```

- Le premier paramètre est la référence sur l'objet,
- Le second définit le type de message. Les différents messages sont définis comme suit:

```
CallBackInformation(HWND, C2DM_GETMAX, (LONG)(LPFPOINT)lpFpoint, 0L)
```

Par ce message, l'objet reçoit la valeur maximale de la courbe dans le buffer pointé par lpFpoint.

- Le troisième paramètre est un pointeur sur une structure de type FPOINT,
- Le quatrième n'est pas utilisé.

```
CallBackInformation(HWND, C2DM_GETMIN, (LONG)(LPFPOINT)lpFpoint, 0L)
```

Par ce message, l'objet reçoit la valeur minimale de la courbe dans le buffer pointé par lpFpoint.

- Le troisième paramètre est un pointeur sur une structure de type FPOINT,
- Le quatrième n'est pas utilisé.

```
lCode = CallBackInformation(HWND, C2DM_GETDATA, (LONG)(LPFPOINT)lpFpoint, lNumber)
```

Par ce message, l'objet reçoit les valeurs d'un point de numéro lNumber dans le buffer pointé par lpFpoint.

- Le troisième paramètre est un pointeur sur une structure de type FPOINT,
- Le quatrième défini le numéro du point,
- Le retour de la fonction est zéro s'il n'y a plus de point, sinon la valeur est positive.

5. Actions effectuées par l'utilisateur.

```
Sélection d'un point ou d'une zone rectangulaire
```

Si le curseur se situe dans la zone graphique (le curseur prend la forme d'une mire), la sélection s'effectue pour un point par un clic souris et pour un rectangle par un clic souris et un déplacement du curseur en maintenant le bouton gauche enfoncé. La sélection peut s'effectuer également par le clavier grâce aux touches de déplacement si la zone graphique possède le focus.

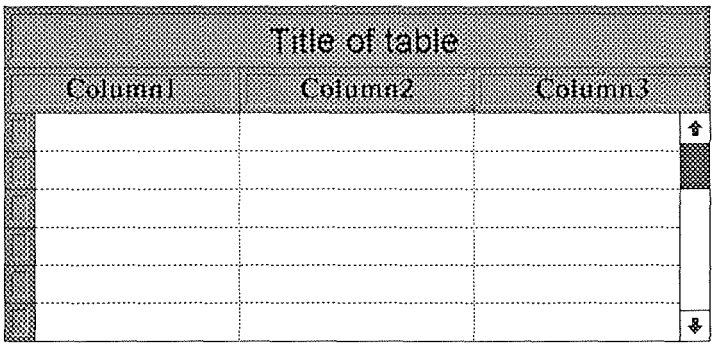
```
Défilement horizontal ou vertical
```

Cette opération est valable si la vue est partielle. Dans ce cas, une ou deux barres de défilement apparaissent. Le défilement s'effectue par un clic dans les différentes zones des barres de défilement, ou par le clavier (touches de déplacement) si la zone graphique possède le focus.

i) Spécification de l'objet interactif de type "Table".

Ce type d'OI que nous voulons définir est sûrement un des OIs qui manque le plus dans un environnement tel que *Windows;* car, dans chaque application développée jusqu'à présent celui-ci était nécessaire.
Dans l'environnement *Windows* nous trouvons la "list Box" qui permet d'afficher une liste de chaînes de caractères. Mais, dès que la liste se compose d'une structure plus complexe cet OI devient très vite limité. De plus, lorsque la liste a une taille relativement élevée, nous sommes confrontés à des problèmes de place mémoire. Enfin l'OI "List" ne permet pas une édition directe des éléments la constituant.
Ainsi, il nous paraît intéressant de disposer d'un OI qui puisse afficher des informations sous forme de table; qu'il ne soit pas limitatif quant à la taille des tables à afficher; qu'il puisse supporter l'édition.
Enfin, cet OI s'appuiera sur les OIs définis plus haut pour le typage des variables.
Au niveau de la présentation, nous pourrons définir un titre général, un titre pour chaque colonne, un numéro pour chaque ligne, le bloquage de plusieurs colonnes en cas de défilement horizontal.

1. Représentation graphique.



2. Attributs de spécialisation de l'objet .

TS_EDIT	Rend les cellules éditables. On pourra bloquer certaines colonnes.
TS_HORLINE	Trace des lignes de séparation horizontale.
TS_NOTITLE	La table sera sans titre.
TS_NOTITLECOL	Les colonnes seront sans titre.
TS_NUMBER	Chaque ligne sera numérotée.
TS_VERLINE	Trace des lignes de séparation verticale.

3. Messages de contrôle.

```
SendMessage(hWnd, TM_ACCEPT, 0, 0L)
```

Ce message va mettre à jour les informations dans la table de l'application.

```
SendMessage(hWnd, TM_DELETE, wCount, lMode)
```

Ce message supprime la ligne de numéro wCount. La suppression est visuelle si lMode est égal à zéro, ou visuelle et dans la table si lMode est positif.

```
SendMessage(hWnd, TM_GETCELL, 0, (LONG)(LPLONG)lpLong)
```

Ce message retourne le numéro de la ligne et de la colonne qui possèdent le focus. Si le numéro de la ligne est négatif alors la colonne est sélectionnée. Si le numéro de la colonne est négatif alors la ligne est sélectionnée.

```
SendMessage(hWnd, TM_INSERT, wCount, lMode)
```

Ce message insère une ligne à la position wCount. Cette insertion est visuelle si lMode est égal à zéro, ou visuelle et dans la table si lMode est positif.

```
SendMessage(hWnd, TM_NOEDITCOLUMNS, wNumber, 0L)
```

Ce message empêche toute édition de la colonne de numéro wNumber.

```
SendMessage(hWnd, TM_REDRAW, wNumber, 0L)
```

Ce message redessine la ligne de numéro wNumber si elle est visible. Si wNumber est négatif alors toutes les lignes visualisées sont redessinées.

```
SendMessage(hWnd, TM_SETCALLBACK, 0, (LONG)(LPPROC)lpProc)
```

Ce message fournit l'adresse de la call back function. Cette fonction permettra de charger la table et d'effectuer des opérations d'insertion ou de suppression.

```
SendMessage(hWnd, TM_SETNUMCOLUMNS, wNumber, 0L)
```

Ce message détermine le nombre de colonnes.

```
SendMessage(hWnd, TM_SETNUMFIXED, wNumber, 0L)
```

Ce message détermine le nombre de colonnes fixes en cas de défilement horizontal.

```
SendMessage(hWnd, TM_SETTEXTCOLUMN, wCount, (LONG)(LPSTR)szTitle)
```

Ce message définit le titre de la colonne de numéro wCount.

```
SendMessage(hWnd, TM_SETTYPECOLUMN, wNumber, lType)
```

Ce message détermine le type de la colonne. lType peut valoir: TEXT, FLOAT, LONG, DATE, TIME.

```
SendMessage(hWnd, TM_SETWIDTHCOLUMNS, wNumber, lWidth)
```

Ce message détermine la largeur de la colonne wNumber.

4. Messages de notification.

```
SendMessage(hWndParent, WM_COMMAND, ID, MAKELONG(TN_CELLCHANGE, hWnd))
```

Ce message notifie la fenêtre parent que la sélection d'une cellule a changé.

Il nous reste à définir le format de la call back function:

```
LONG FAR PASCAL CallBackInformation(HWND, WORD, WORD, LONG, LONG)
```

- Le premier paramètre est la référence sur l'objet,
- Le second définit le type de message. Les différents messages sont définis comme suit:

```
CallBackInformation(HWND, TM_DEL, wPos, 0L, 0L)
```

Par ce message, l'objet envoie un ordre de suppression à la position wPos au niveau de la table de l'application.

- Le troisième paramètre définit la position de suppression,
- Le quatrième n'est pas utilisé,
- Le cinquième n'est pas utilisé.

```
CallBackInformation(HWND, TM_GETDATA, wSize, (LONG)(LPTYPE)lpBuffer, lPos)
```

Par ce message, l'objet envoie la valeur d'une cellule de position lPos dans le buffer pointé par lpBuffer de taille wSize.

- Le troisième paramètre définit la taille du buffer,
- Le quatrième est un pointeur sur une structure qui dépend du type,
- Le cinquième définit la position de la cellule.

CallBackInformation(HWND, TM_GETNUMBER, 0, (LONG)(LPLONG)lpNumber, 0L)

Par ce message, l'objet reçoit le nombre maximal de lignes de la table dans le buffer pointé par lpNumber.

- Le troisième paramètre n'est pas utilisé,
- Le quatrième est un pointeur sur un long,
- Le cinquième n'est pas utilisé.

CallBackInformation(HWND, TM_INS, wPos, 0L, 0L)

Par ce message, l'objet envoie un ordre d'insertion à la position wPos au niveau de la table de l'application.

- Le troisième paramètre définit la position d'insertion,
- Le quatrième n'est pas utilisé,
- Le cinquième n'est pas utilisé.

CallBackInformation(HWND, TM_SETDATA, 0, (LONG)(LPTYPE)lpBuffer, lPos)

Par ce message, l'objet reçoit la valeur d'une cellule de position lPos dans le buffer pointé par lpBuffer.

- Le troisième paramètre n'est pas utilisé.
- Le quatrième est un pointeur sur une structure qui dépend du type.
- Le cinquième définit la position de la cellule.

5. Actions effectuées par l'utilisateur.

Sélection d'une cellule.

La sélection s'effectue par un clic souris ou par un déplacement via les touches de déplacement.

Edition d'une cellule (si mode Edition)

(Voir action pour les sous-classes et classe Edit)

Sélection d'une ligne

La sélection sera effectuée par un clic souris dans la case gauche ()correspondant à la ligne. Ce type de sélection est utile pour faire de la suppression ou de l'insertion.

Sélection d'une colonne

La sélection sera effectuée par un clic souris dans le titre de la colonne. Ce type de sélection peut être utile pour le "Copy/Paste"

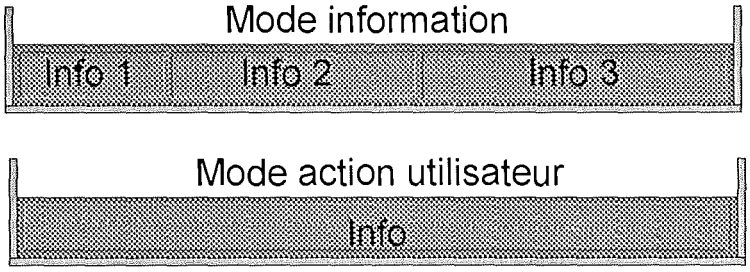
Défilement horizontal ou vertical

(Cfr barre de défilement)

j) Spécification de l'objet interactif de type "StateLine".

Ce type d'objet est rencontré fréquemment dans les applications de type Multiple Document Interface[14] (MDI). Il se présente comme une ligne située en bas de la fenêtre principale où une série d'informations seront affichées. Ces informations sont de deux types. Les premières concernent certains paramètres de l'environnement (Heure, Touche numlock enfonçée, Touche de Majuscule enfonçée, ...). Les deuxièmes concernent les actions que l'utilisateur effectue au niveau des menus et des commandes. La taille de cet objet est définie automatiquement. De plus, comme cet objet est inséré dans une fenêtre principale et non dans une boîte de dialogue, il ne devra pas être intégré dans les outils de type éditeurs de boîtes de dialogue.

1. Représentation graphique.



2. Attributs de spécialisation de l'objet .

Pas de styles particuliers.

3. Messages de contrôle.

SendMessage(hWnd, SL_ADJUST, 0, 0L)

Ce message réajuste la position et/ou la taille de l'objet lorsque la fenêtre parent change de position ou de taille.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième n'est pas utilisé,
- Le quatrième n'est pas utilisé,

[14]Le MDI est une application qui peut avoir plusieurs fenêtres en même temps. Celles-ci sont créent dynamiquement (Attention, ces fenêtres ne sont pas des boîtes de dialogue).

SendMessage(hWnd, SL_DEFINE, wSize, (LONG)(LPUDEFTAB)lpUDefTab)

Ce message fournit à l'objet un tableau qui définit les différentes cases d'affichages. Le nombre de cases est donné par wSize.
La structure est composée de deux champs. Le premier définit la position en absolu du début de la case en caractères. Le deuxième définit la longueur de la case en caractères.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit le nombre de cases,
- Le quatrième est un pointeur sur un tableau de valeurs définies par le type DEFTAB.

SendMessage(hWnd, SL_SETTEXTPOS, wPos, (LONG)(LPSTR)szText)

Ce message affiche dans la case de numéro wPos une chaîne de caractères pointée par szString.
Pour l'affichage d'une chaîne qui correspond à une action utilisateur, on utilise le message WM_SETTEXT.

- Le premier paramètre est l'objet interactif,
- Le deuxième est le type de message,
- Le troisième définit le numéro de la case où doit être affiché le texte,
- Le quatrième est un pointeur sur une chaîne de caractères.

4. Messages de notification.

Pas de messages de notification.

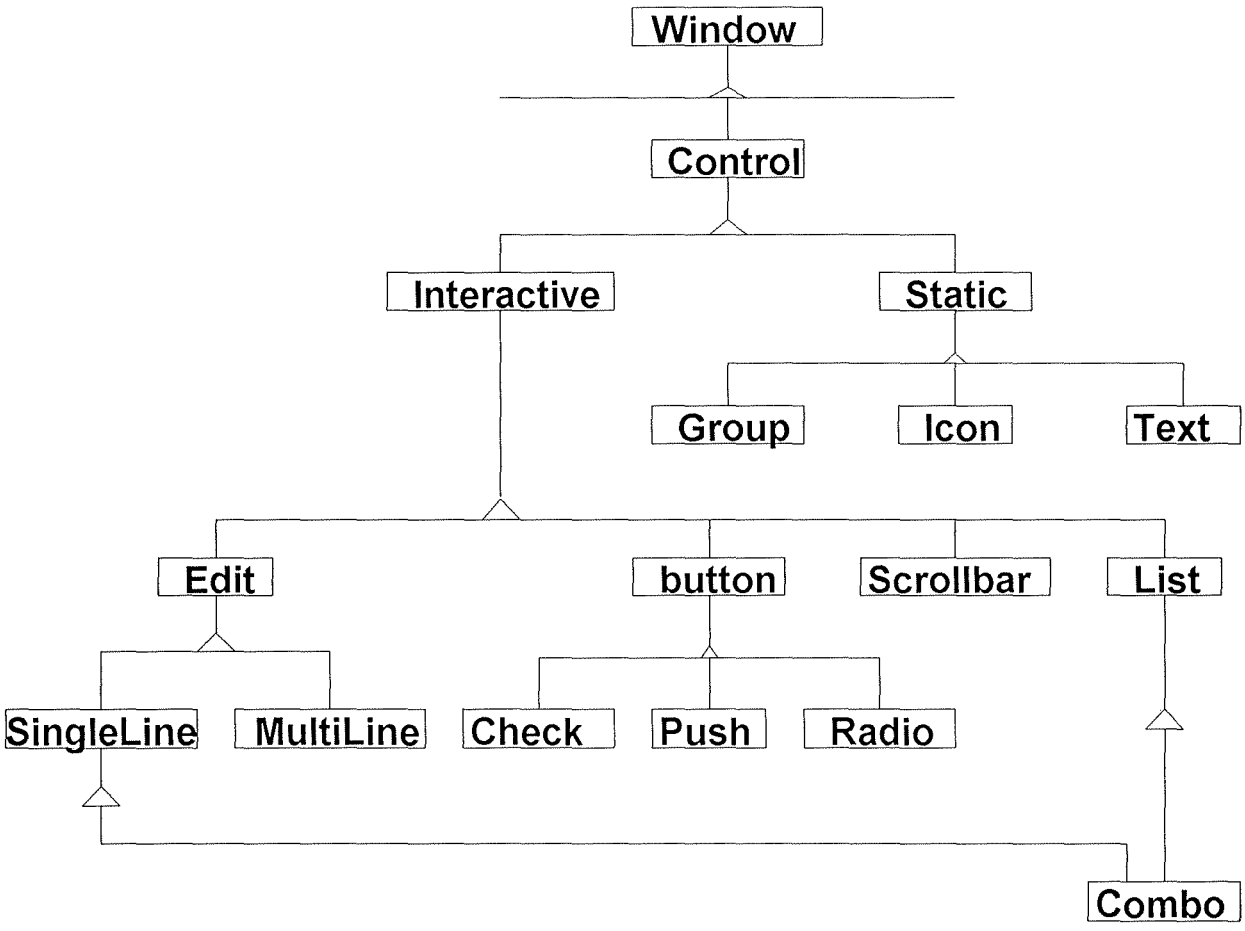
5. Actions effectuées par l'utilisateur.

Pas d'actions utilisateur.

V. Conception des composants.

a) Modélisation de l'existant.

Avant de nous lancer dans la conception des différents Objets Interactifs, il nous a paru intéressant de définir le graphe d'héritage des objets interactifs présent dans l'environnement *Windows*. Ce graphe d'héritage ne sera qu'une vue partielle des objets présents dans l'environnement. De plus nous nous limiterons à l'énumération des classes sans entrer dans le comportement de celles-ci. Enfin notons que cette hiérarchisation n'est pas unique. Pour s'en convaincre il suffit de consulter la littérature.



Notons également que ce graphe d'héritage est purement conceptuel car dans l'environnement *Windows*, il n'est pas présent quant à son implémentation.
La classe Control hérite de la classe Window qui définit le comportement général de toute fenêtre. La classe Control définit le comportement que tous les objets interactifs peuvent avoir en commun, c'est-à-dire leur intégration dans les boîtes de dialogue et dans les outils de construction de celles-ci. La classe Interactive est une classe qui définit les objets qui ont une interaction avec l'utilisateur tandis que la classe Static

définit le comportement pour les classes qui n'ont aucune interaction avec l'utilisateur. La classe Text permet l'affichage d'une chaîne de caractères. La classe Icon affiche une icône. La classe Group permet d'effectuer un groupement logique de plusieurs objets. Notons que dans l'environnement *Windows*, l'objet de type Group est une spécialisation de la classe Button. La classe Edit permet l'édition d'une chaîne de caractères sur une seule ligne (SingleLine) ou sur plusieurs (MultiLine). La classe Button peut être, suivant la spécialisation ,un bouton pour effectuer une action, une case à cocher ou une case radio (exclusive). La classe List offre la possibilité d'une sélection simple ou multiple d'items. Enfin, la classe Combo offre l'édition et/ou la sélection dans une liste.

b) Conception de l'OI "Meter".

1. Analyse.

Cet OI sera défini comme étant une classe de type statique.
Pour mémoriser la borne maximale et la valeur courante, nous avons besoin de deux valeurs qui seront de type WORD.

- maximumValue : WORD = 100
- currentValue : WORD = 50

Les valeurs par défaut sont nécessaires pour avoir une vision réaliste de l'objet interactif lors de sa gestion dans l'éditeur de boîtes de dialogue.

Comme méthodes publiques associées à cette classe, nous trouverons:

- SetMaximumValue(maximumValue: WORD) : INT
- SetCurrentValue(currentValue: WORD) : INT
- GetMaximumValue() : WORD
- GetCurrentValue() : WORD

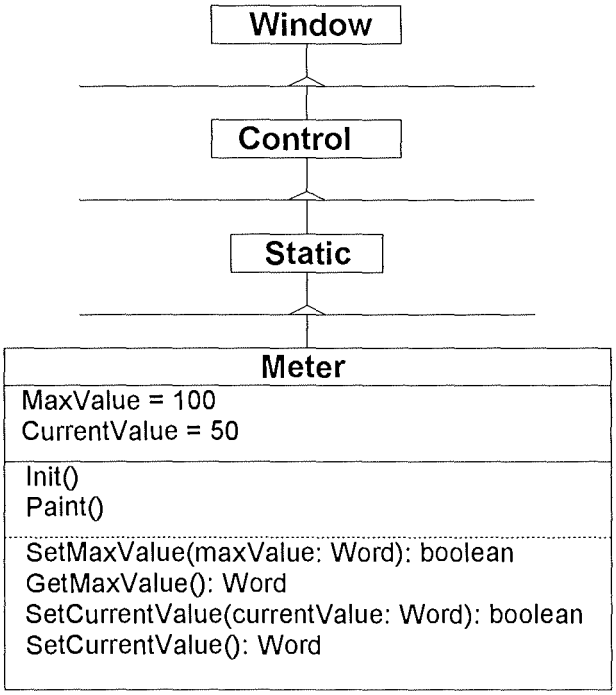
Ces différentes méthodes proviennent directement de la spécification de l'OI

Comme méthodes privées associées à cette classe, nous aurons:

- Initialisation()
- Paint()

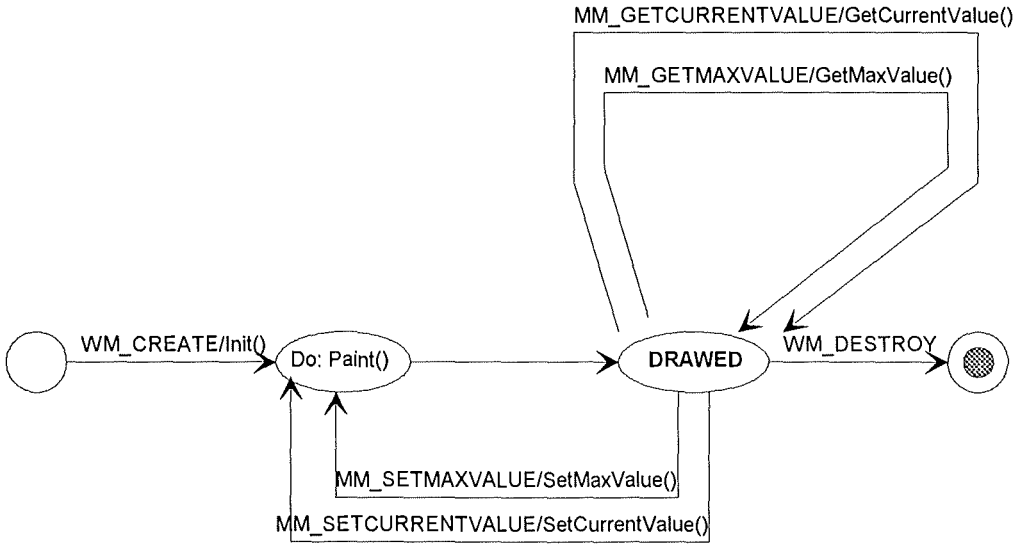
Ces méthodes sont déduites du modèle de la dynamique et du modèle fonctionnel.

- *Modèle Objet :*



Par rapport au graphe d'héritage vu précédemment nous nous proposons de placer l'OI Meter comme sous-classe de la classe Static car, conceptuellement, cet objet n'est pas une spécialisation d'un objet existant et il n'a pas d'interaction avec l'utilisateur.

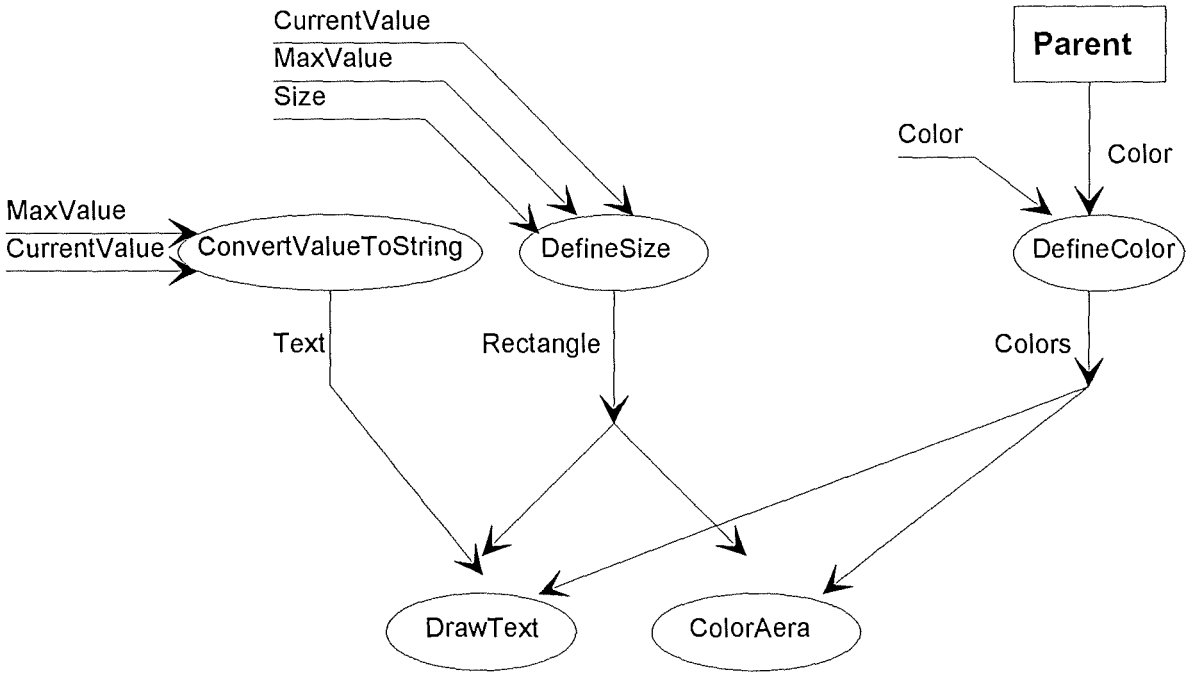
- *Modèle de la dynamique*[15] *:*



[15]Etant donné que les messages ont un format unique, les attributs des évènements ne sont pas indiqués.

Lorsqu'une instance d'une classe est créée, elle reçoit d'abord le message WM_CREATE qui permet d'initialiser certains paramètres. Dans notre cas, de façon a avoir une représentation réaliste de l'objet lors de sa manipulation dans les éditeurs de boîtes de dialogue, nous allons déterminer la valeur maximale et la valeur courante. De cette façon, il faut définir une nouvelle méthode d'instance qui sera privée.
Suite à ce message ou aux messages de modification d'états, il faut dessiner la présentation de l'objet. C'est ainsi que nous définissons une nouvelle méthode d'instance pour dessiner l'objet. Les autres opérations sont déduites de la spécification.

- *Modèle fonctionnel de la méthode "Paint()" :*



Ce shéma définit les opérations de plus bas niveau de la méthode "Paint()". Nous pouvons voir ces opérations comme des appels à des fonctions présentes dans l'environnement *Windows* ou dans des librairies spécialisées.

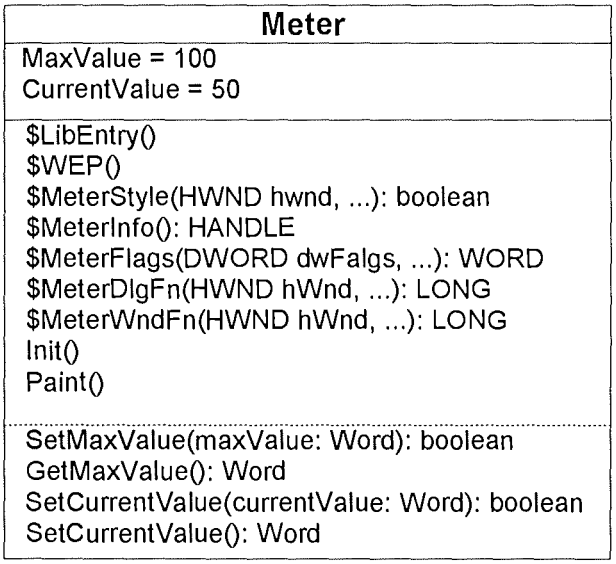
2. Design du système.

Jusqu'ici nous n'avons pas encore parlé du problème d'intégration de cet objet dans l'environnement Windows et dans le(s) outil(s) de conception de boîtes de dialogue. Pour y parvenir, nous allons devoir intégrer l'objet dans une librairie de type DLL et définir un protocole qui soit connu de l'environnement. Ce protocole sera défini dans la phase de design de la classe. Pour cette phase, on se limitera donc à définir un module qui, au niveau de l'implémentation, se traduit par le concept de librairie dynamique (DLL).

3. Design de la classe.

Comme nous venons de le voir, nous devons définir le protocole pour l'intégration de l'objet dans l'environnement.
Afin que cette nouvelle classe soit reconnue, il faut la répertorier dans l'environnement. Cette opération est effectuée par une fonction standard qui doit être intégrée dans le code. Elle porte le nom de LibEntry().
De même, une fonction standard doit être intégrée pour supprimer la liaison entre cet objet et l'environnement. Elle porte le nom de WEP().
Pour intégrer l'objet dans des outils de conception de boîtes de dialogue, il faut définir quatre fonctions qui ont les fonctionnalités suivantes:
La première fonction permet de retrouver le nom de la classe et les styles qu'une instance doit avoir lors de sa création.
La deuxième fonction donne accès à la fonction qui permet de faire une spécialisation de l'objet.
La troisième fonction permet de construire une chaîne de caractères qui définit l'instance. Cette chaîne est écrite dans le fichier de ressources associées à la boîte de dialogue.
La quatrième fonction est un appel à une boîte de dialogue qui permet de paramétriser l'instance qui doit y être intégrée.
Enfin, comme cette classe hérite des propriétés de la classe Window, il faut définir une fonction qui intègre le comportement de l'objet.

En fait, nous pouvons voir ces fonctions comme des méthodes de classe.



4. Implémentation.

(Voir annexe)

c) Conception de l'OI "EditLong".

1. Analyse.

Cet OI sera défini comme étant une classe de type interactif.
Pour mémoriser la borne minimale, maximale et la valeur courante, nous avons besoin de trois valeurs qui seront de type Long:

- minimumValue: Long = - 2^{31}
- maximumValue: Long = 2^{31} - 1
- currentValue: Long

Comme méthodes publiques associées à cette classe, nous trouverons:

- SetMaximumValue(maximumValue: LPLONG) : LONG
- SetMinimumValue(minimumValue: LPLONG) : LONG
- SetCurrentValue(currentValue: LPLONG) : CODE
- GetMaximumValue(size: WORD, maximumValue: LPLONG) : CODE
- GetMinimumValue(size: WORD, minimumValue: LPLONG) : CODE
- GetCurrentValue(size: WORD, currentValue: LPLONG) : CODE
- CheckCurrentValue() : CODE

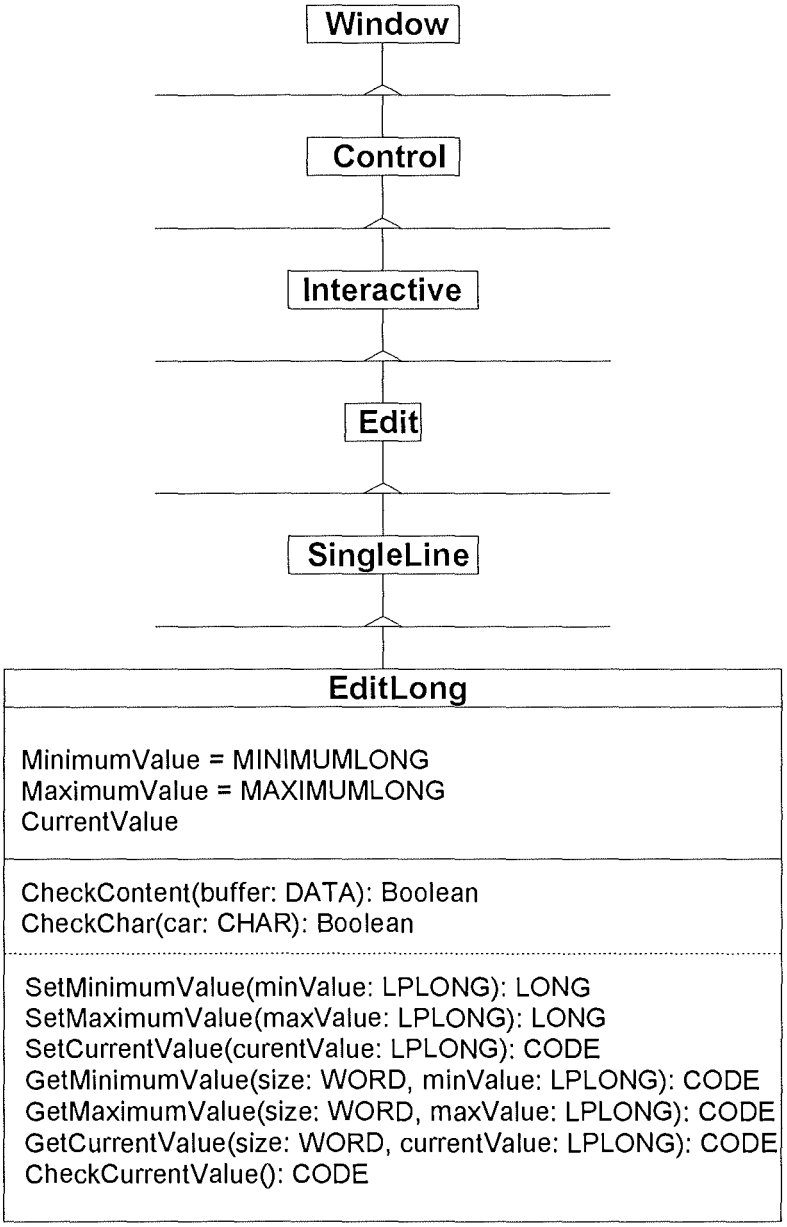
Ces différentes méthodes proviennent directement de la spécification de l'OI.

Comme méthodes privées associées à cette classe, nous aurons:

- CheckChar(car: CHAR) : Boolean
- CheckContent(buffer: DATA) : Boolean

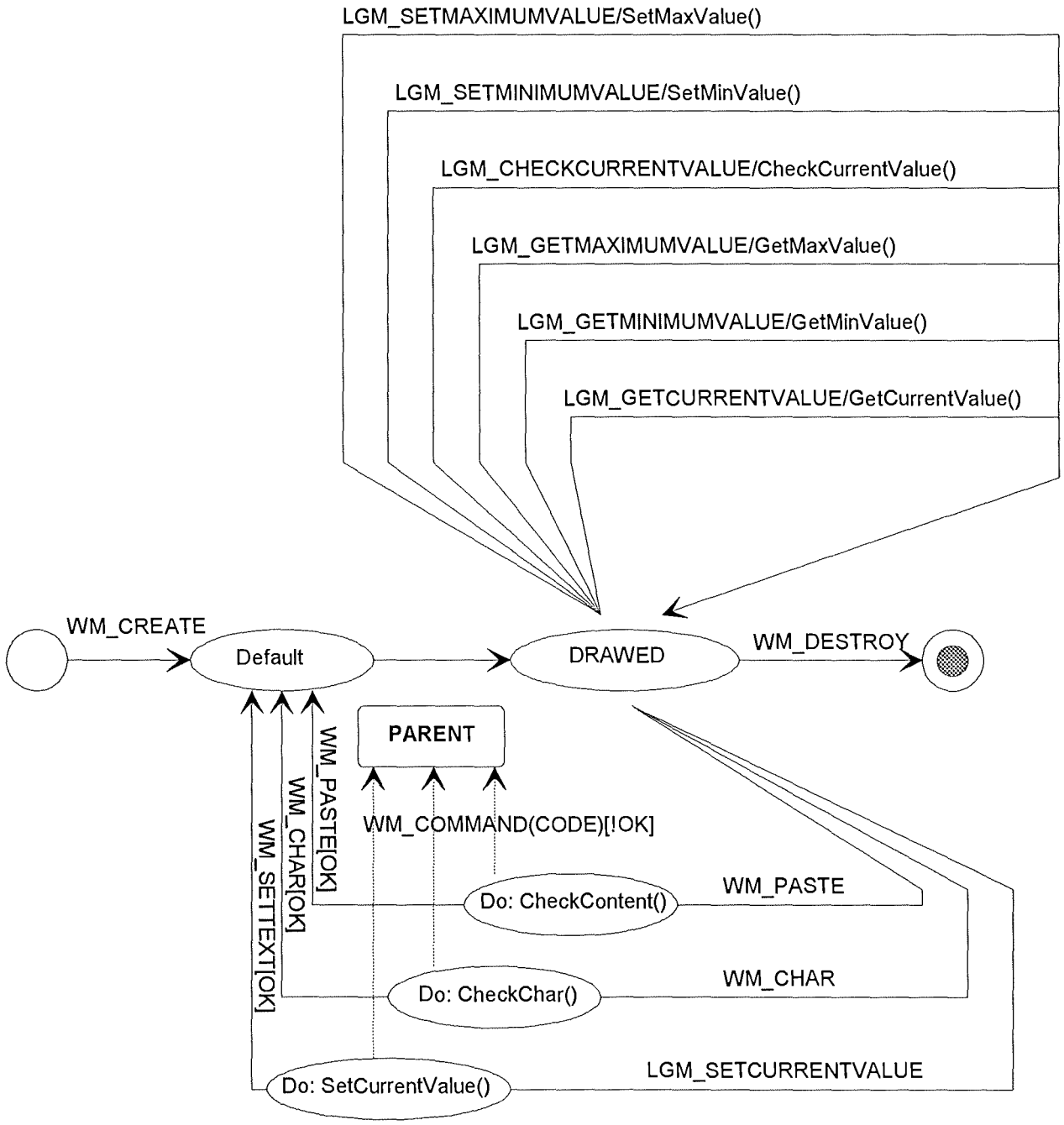
Ces méthodes sont déduites du modèle de la dynamique et du modèle fonctionnel.

- *Modèle Objet :*



Cette classe hérite du comportement de la classe "SingleLine". En effet, conceptuellement, cet OI est une spécialisation de l'OI "SingleLine".

- *Modèle de la dynamique :*



Dans ce schéma, l'état nommé "Default" définit le comportement de la super-classe (SingleLine). Lors de l'entrée d'une valeur incorrecte, l'objet envoie un message de notification à la fenêtre parent. Les opérations "CheckChar()" et "CheckContent()" sont déduites des actions de l'utilisateur. En effet, celui-ci peut entrer des caractères qui sont incorrectes (fonction du type et des bornes) ou utiliser le mécanisme du "Copier/Coller" qui lui peut entraîner une erreur de type ou de borne également. Les autres opérations sont déduites de la spécification.

- *Modèle fonctionnel :*

  Entant donné que la plupart des opérations font appel à des fonctions de niveau inférieur, nous nous limiterons aux deux premiers modèles pour concevoir cet objet.

2. Design du système.

Les considérations que nous avons eues pour la classe "Meter" restent d'application. Le fait que cette classe hérite de la classe "Edit" ne change pas de l'encapsulation dans un module unique (DLL).

3. Design de la classe.

Les méthodes de classe que nous avons définies pour l'objet précédent doivent être adaptées. Dans *Windows*, l'héritage ne se gère pas automatiquement; de ce fait, le code va intégrer des fonctionnalités suplémentaires. Celles-ci seront définies pour gérer l'accès aux informations et au comportement de la super-classe.

4. Implémentation.

(Voir annexe)

d) Conception de l'OI "GraphicButton".

1. Analyse.

Cet OI sera défini comme étant une classe de type interactif.
Pour mémoriser l'image (bitmap) nous avons besoin d'un handle. Pour connaître l'état interne du bouton (bouton enfoncé ou pas, bouton sélectionné par la souris, ...) nous avons besoin d'un WORD.

- hBitmap : HANDLE = NULL
- wPrivate : WORD = 0

Comme méthodes publiques associées à cette classe, nous trouverons:

- GetCheck() : Boolean
- SetCheck(bSelect: Boolean)
- SetStyle(wStyle: WORD)

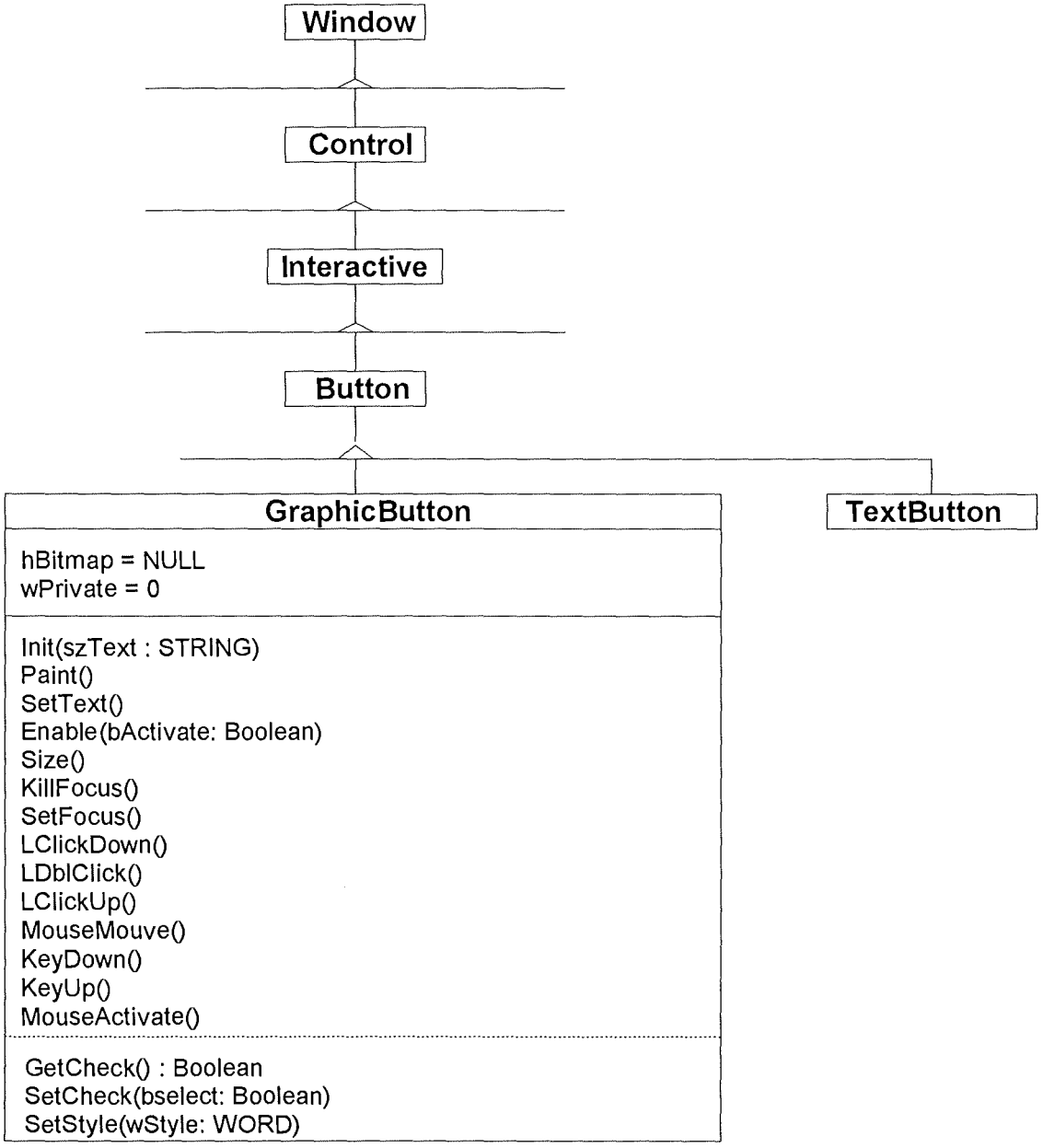
Ces différentes méthodes proviennent directement de la spécification de l'OI.

Comme méthodes privées associées à cette classe, nous aurons:

- Init()
- paint()
- SetText(szText: STRING)
- MouseActivate()
- MouseMouve()
- SetBDown()
- SetBUp()
- SetKDown()
- SetKUp()
- SetFocus()
- KillFocus()
- Enable(bActivate: Boolean)
- Size()

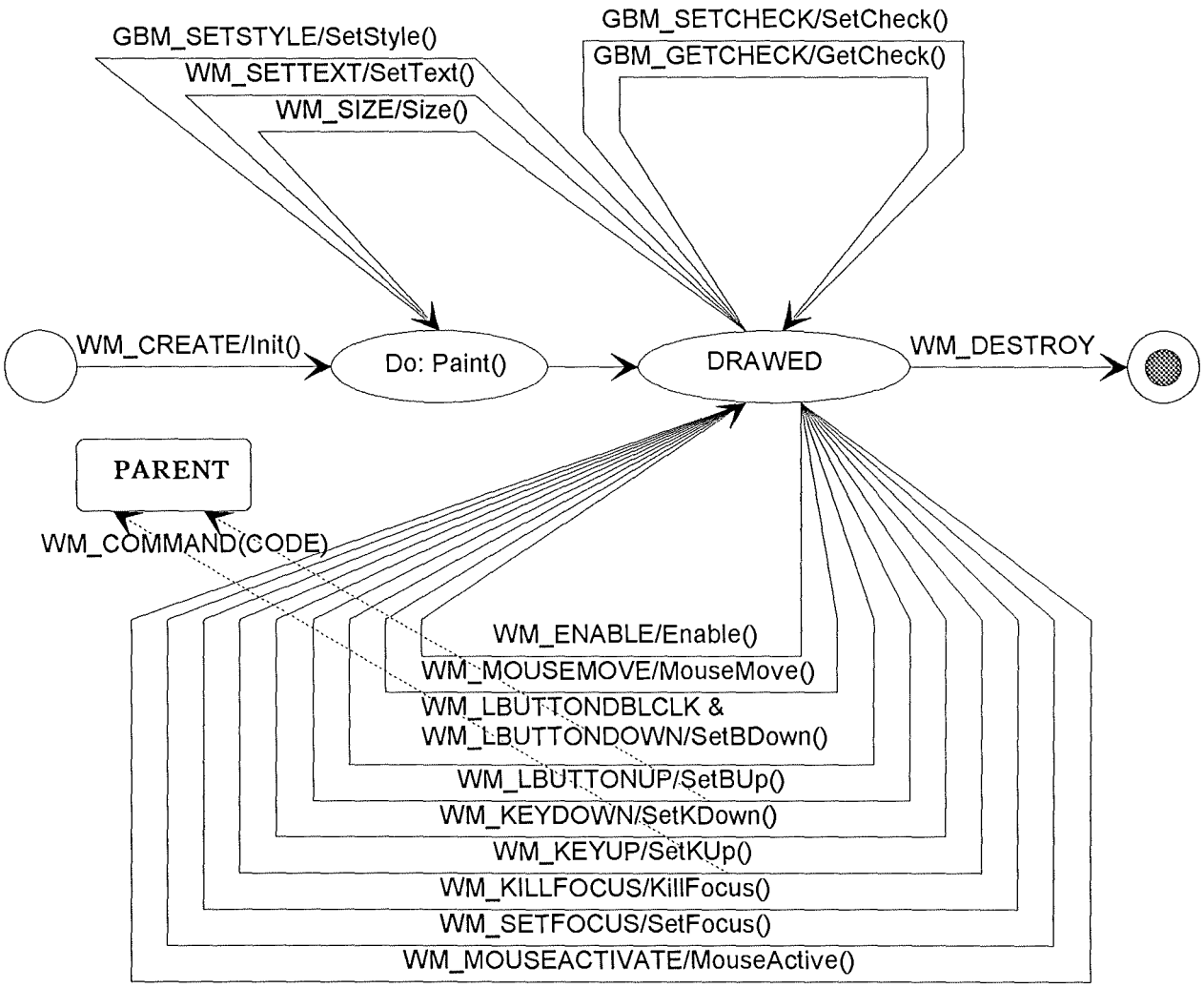
Ces méthodes sont déduites du modèle de la dynamique et du modèle fonctionnel.
Notons que la plupart des méthodes privées sont des redéfinitions des méthodes héritées de la classe Window.

- *Modèle objet :*



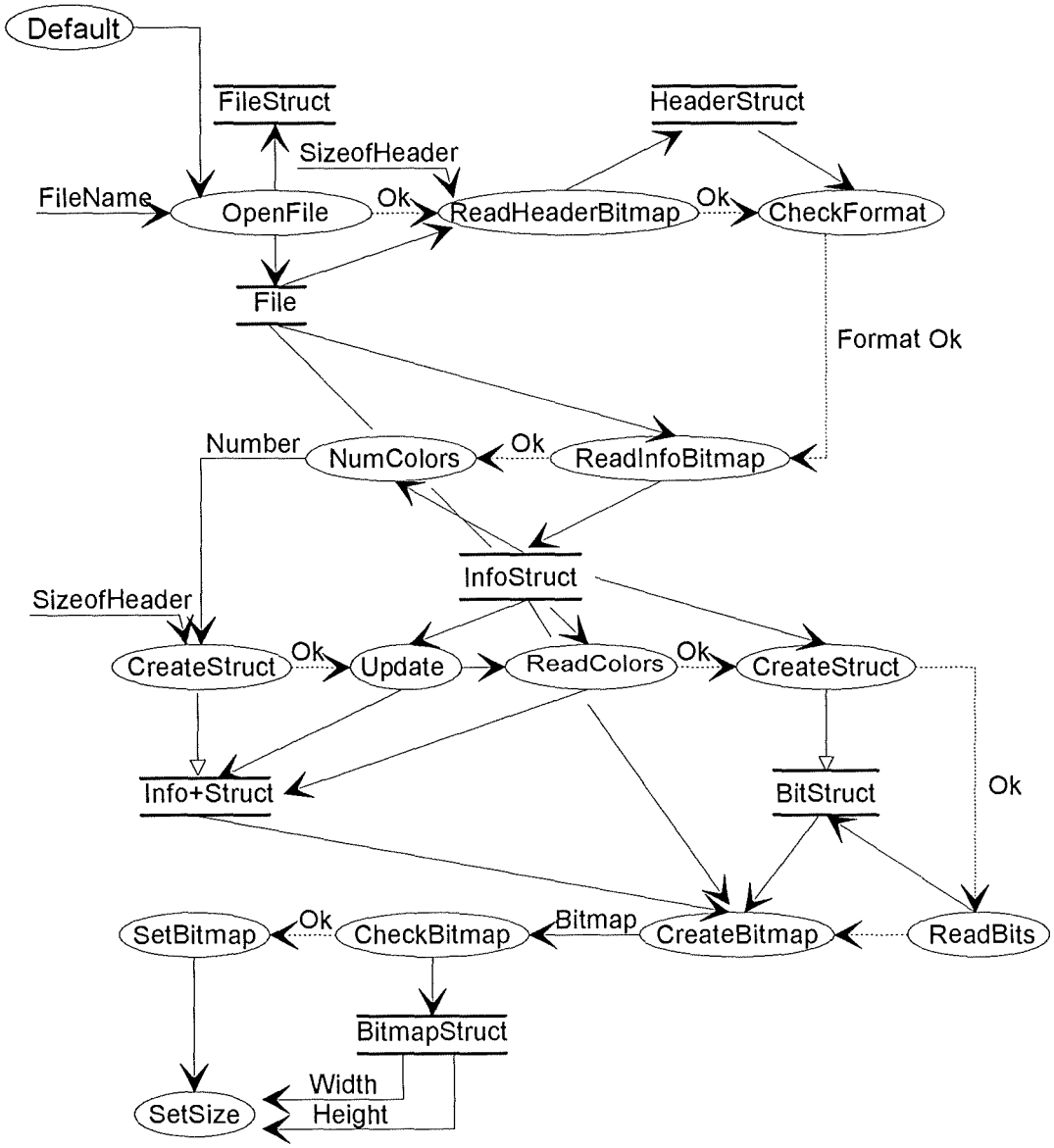
Etant donné que le bouton graphique et le bouton standard (texte) ont le même comportement, il nous semble intéressant de définir une classe abstraite de type bouton qui soit la super-classe des deux classes citées ci-dessus. Rappelons que cette hiérarchie est purement conceptuelle.

- *Modèle de la dynamique :*



Comme nous le voyons sur le schéma, on retrouve les opérations qui ont été définies dans la spécification. De plus nous trouvons des méthodes qui sont déduites des actions de l'utilisateur et des actions que nous appelerons secondaires ou naturelles ("MouseMove()" en est un exemple). Enfin, certaines méthodes héritées sont redéfinies ou spécialisées (l'opération "Enable()" doit être spécialisée afin qu'elle "grise" le bouton). Notons tout de même que certaines méthodes peuvent être ajoutées suite à des tests au niveau de l'implémentation (il en est ainsi pour la méthode "MouseActivate()" qui a été introduite pour différencier le cas d'une activation par la souris ou par le clavier).

- *Modèle fonctionnel de l'opération SetText() :*



Comme nous le voyons sur le schéma, cette opération effectue la lecture d'un fichier de type "bitmap" et de nom "FileName" afin d'initialiser une structure de type Bitmap. Cette opération est une spécialisation de la fonction héritée car on exécute d'abord cette dernière.

2. Design du système.

(Voir la classe Meter)

3. Design de la classe.

(Voir la classe Meter)

4. Implémentation.

(Voir annexe)

e) Conception de l'OI "StateLine".

1. Analyse.

Cet OI sera défini comme étant une classe de type statique.
Pour mémoriser le tableau qui définit les différentes cases nous avons besoin d'un HANDLE, un WORD pour stocker le nombre de cases, ainsi que d'un WORD pour stocker certains états (mode information ou action).

- hTab: HANDLE = NULL
- wNumber: WORD = 0
- wPrivate : WORD = 0

Comme méthodes publiques associées à cette classe, nous trouverons:

- Define(wNumber: WORD, lTab: LPDEFTAB)
- Adjust()
- SetTextPos(wPos: WORD, lText: LPSTR)
- SetText(lText: LPSTR)

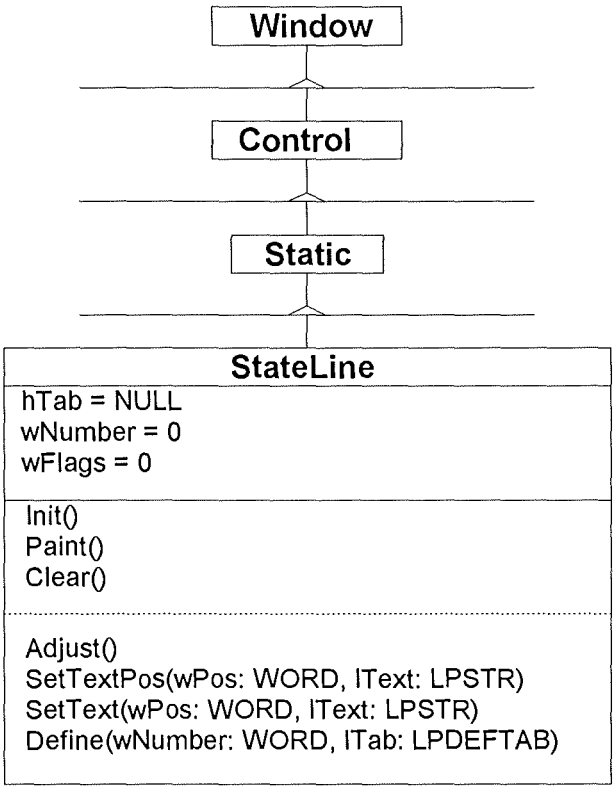
Ces différentes méthodes proviennent directement de la spécification de l'OI.

Comme méthodes privées associées à cette classe, nous aurons:

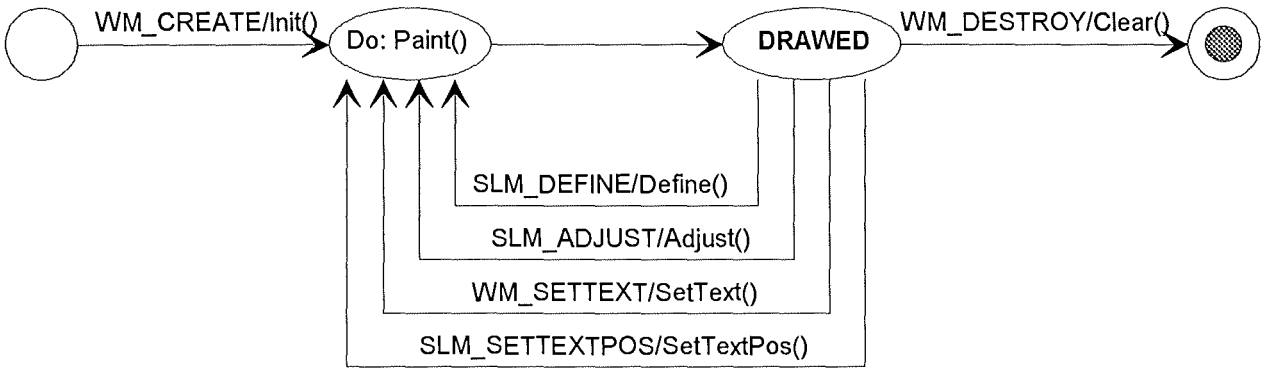
- Init()
- paint()
- Clear()

Ces méthodes sont déduites du modèle de la dynamique et du modèle fonctionnel.
Notons que la méthode SetText est une redéfinition de la méthode héritée de la classe Window.

- *Modèle objet :*



- *Modèle de la dynamique :*



Les opérations dérivent directement de la spécification. Seules les fonctions "Init() et "Clear()" sont définies de façon à creer des structures dynamiques.

- *Modèle fonctionnel.*

La plupart des opérations font appel à des fonctions graphiques d'un niveau inférieur.

2. Design du système.

Comme nous l'avons défini lors de la spécification, cet objet ne devra pas être intégré dans les différents éditeurs de boîtes de dialogue car cet objet est toujours associé à une fenêtre principale.

3. Design de la classe.

Pour cette classe, il n'y aura pas de fonctions d'intégration au niveau des éditeurs de boîtes de dialogue.

4. Implémentation.

(Voir annexe)

f) Conception de l'OI "Curve2D".

1. Analyse.

Cet OI sera défini comme étant une classe de type interactif.
Comme nous l'avons vu au niveau de la spécification, cet objet va être composé d'un ensemble d'objets spécialisés. Avec cette approche, quelques problèmes se posent. Etant donné que ces objets vont devoir dialoguer dans un environnement événementiel, qu'ils sont tous des "fils" d'une fenêtre parent, et qu'il n'existe pas de mécanisme standard pour définir des relations entre objets de même niveau, nous allons devoir définir une structure qui permette ce dialogue inter-objets. Ainsi, chaque objet qui devra dialoguer avec des objets de même niveau possèdera, en interne, une référence sur eux.
Avant de se lancer dans l'analyse, il nous a semblé utile de définir un prototype pour valider certains concepts. Comme nous venons de le voir, le problème réside essentiellement dans la communication inter-objets. Nous avons défini les objets et leur protocole de communication sans entrer dans les problèmes d'affichage. Etant donné que les tests effectués n'ont pu être mené jusqu'au bout, nous ne pouvons affirmer que ce mécanisme est applicable.

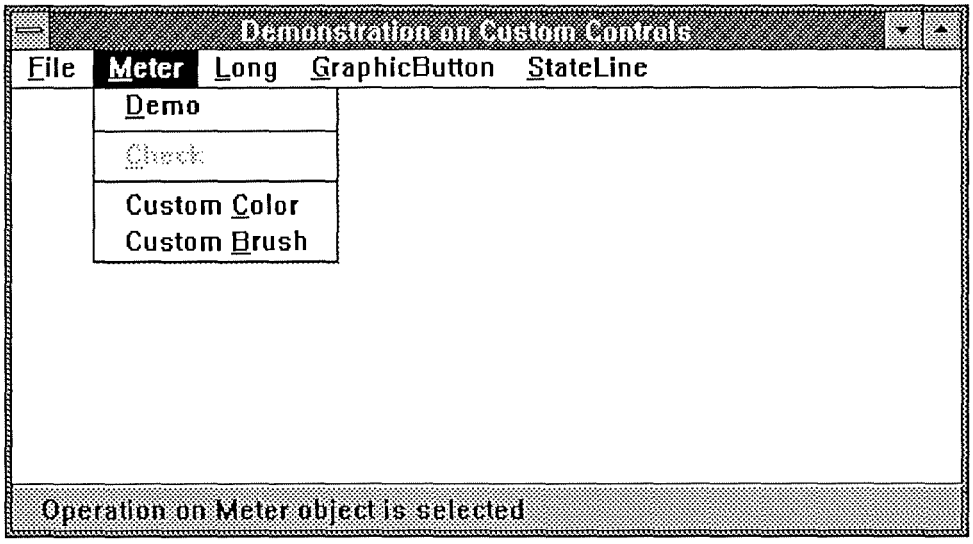
VI. Impact de la conception sur l'implémentation.

Comme nous l'avons vu dans l'introduction, l'implémentation a été effectuée en langage C. Malgré le fait que celui-ci ne soit pas orienté objet, la transposition n'a pas posé de problème. Une des raisons majeures est que l'on retrouve plusieurs concepts de l'orienté objet dans l'environnement *Windows*.
Notons tout de même que toutes les ressources disponibles dans l'environnement *Windows,* à l'exeption des "fenêtres", sont accessibles via des appels de fonction (pas de notion d'objet). Nous devons voir ces ressources comme des modules (librairies) d'un niveau inférieur. Dans la structure en couche (niveau) exposée au cours de Méthodologie de Développement de Logiciels de deuxième licence (MDL), nous placerions ces librairies au niveau ②. De plus, malgré le fait que nos composants sont considérés comme des "applications" dans l'environnement Windows, ils seront placés au niveau ⑥. La justification est que ces composants sont des objets réutilisables qui n'ont aucune existence en tant qu'applications.

VII. Conception d'une mini-application.

Cette application mettra en oeuvre les différents objets interactifs qui ont été implémentés. Le but de cette application est double: d'une part ,faire la démonstration du comportement des objets et, d'autre part, en tester la validité.
Nous n'allons pas faire l'analyse de cette application tant le problème est simple. En effet, l'application n'est, en fait, qu'une mise en oeuvre des objets interactifs en leur envoyant quelques messages.

Nous trouverons ci-dessous une vue de cette application.



VIII. Conclusions et Perspectives.

Ce travail avait pour objectif de spécifier et développer des objets interactifs pour l'environnement *Windows* en utilisant une méthode de conception orientée objet. Seul le langage de programmation était imposé (langage C).

Nous trouvons ci-dessous la liste des objets envisagés:

Curve2D	Permet l'affichage de courbes, iso-courbes.
EditLong	Permet l'acquisition d'une valeur de type LONG.
EditFloat	Permet l'acquisition d'une valeur de type FLOAT.
EditDateTime	Permet l'acquisition d'une valeur de type DATE ou TIME.
EditMask	Permet l'acquisition d'une valeur contrainte par un masque de saisie.
GraphicButton	Equivalent de l'objet standard de type Button mais dont la représentation sémantique est graphique.
Meter	Permet de visualiser l'état d'avancement d'une tâche.
StateLine	Permet l'affichage d'informations dans une fenêtre principale.
Table	Permet l'affichage et l'édition d'une table.

Comme plusieurs problèmes ont été abordés, nous allons scinder la conclusion en plusieurs parties chacune d'elle couvrant un aspect du problème.

Les résultats par rapport aux objectifs.

Comme nous l'avons annoncé dans l'introduction, seule une partie des objets interactifs a pu être développée. Les objectifs sont partiellemement atteints du fait que certains problèmes n'ont pu être solutionnés qu'après maintes recherches. Les objets intégralement implémentés sont au nombre de quatre (Meter, EditLong, GraphicButton et StateLine). Le cinquième (Curve2D) n'a pu être développé en raison de l'approche qui a été adoptée et seul un prototype a été élaboré. Les autres objets ont seulement été spécifiés.

Intérêt de ce type de développement.

Les objets implémentés dans le cadre de ce travail sont déjà intégrés dans des applications qui sont en cours de développement dans la société *MTS*. L'intérêt essentiel est le gain de temps. Si tous les objets étaient développés, la construction d'une interface se limiterait essentiellement à un assemblage de composants.

L'apport de la méthode Object Modeling Technique (OMT).

De par la nature un peu particulière de ce type de développement, nous constatons que cette méthodologie n'est peut être pas toujours adaptée. En effet, hormis certains objets qui sont en fait des aggrégats d'autres objets, la communication inter-objets n'a pas été exploitée. Enfin, en raison de la nature de l'environnement *Windows*, il aurait été préférable d'adopter une approche par prototypage. En effet, que ce soit pour explorer de nouvelles techniques ou pour connaître les réactions de l'environnement suite à l'envoi de certains messages, la technique du prototypage nous semble adéquate (temps et complétude). De plus, sans le support d'un outil CASE on s'aperçoit rapidement que l'on fait plus du dessin que de la conception. Cet outil serait, dans un premier temps, une aide à la modélisation graphique et dans un deuxième temps, une aide à la conception et à la génération de codes. Ainsi, seuls les modèles objet et de la dynamique ont été essentiellement exploités.

L'implémentation des différents composants.

C'est ici que la plupart des problèmes ont été rencontrés. Le fait d'avoir développé dans l'environnement *Windows* pendant un an et demi ne nous a pas toujours été d'un grand secours. La cause en est que pour concevoir des objets interactifs qui s'intègrent dans un environnement, il faut avoir une bonne connaissance des mécanismes internes des objets standards. En outre, les informations concernant le développement d'OIs dans *Windows* sont plutôt rares.
Les classes développées bénéficient de certains concepts objets présent dans l'environnement or, il n'en est pas de même pour les autres ressources. Ces dernières devant être considérées comme des fonctions d'un niveau inférieur (structuration en niveau).

Extensions possibles.

Il est intéressant de parler des extensions possibles que l'on pourrait apporter aux objets développés. Si nous prenons l'exemple du bouton graphique, il conviendrait de lui adjoindre la possibilité d'afficher un texte et un raccourci clavier. Au niveau des sous-classes de la classe Edit il serait utile de leur joindre la possibilité d'un appel à une call back function pour valider la donnée entrée. Enfin, nous aurions pu également spécialiser la classe Combo[16] de la même façon que nous l'avons fait pour la classe Edit.

---

[16]Pour rappel, la classe Combo hérite du comportement de la classe Edit et/ou de la classe List. Elle permet l'édition et/ou la sélection dans une liste.

Considérations générales.

Nous assistons, à l'heure actuelle, à une petite révolution dans le monde informatique et en particulier dans le domaine des interfaces utilisateurs. Il nous paraît donc essentiel de s'intéresser aux développements d'outils qui améliorent tant le dialogue avec l'utilisateur (diminution du fossé sémantique, approche par manipulation directe et normalisation) que les temps de développement pour le concepteur. Nous avons l'impression que ce type de développement peut contribuer à diminuer le fossé sémantique et le temps de développement de l'interface utilisateur.

Bibliographie

Développement sous Windows.

Microsoft Windows SDK Reference - Volume 1 (- Microsoft)

Microsoft Windows SDK Reference - Volume 2 (- Microsoft)

Microsoft Windows SDK User's Guide (- Microsoft)

Microsoft Windows SDK Guide to Programming (- Microsoft)

Microsoft Windows SDK Tools (- Microsoft)

Programming Windows 3.0 (Charles Petzold - Microsoft Press)

Windows 3.0: A Developer's Guide (Jeffrey M.Richter - Prentice Hall)

Microsoft Windows Developer's Workshop (Richard Wilton - Microsoft Press)

Developing Windows 3 Applications with Microsoft SDK (Brent Rector - Sams)

Graphics Programming Under Windows (Brian Myers & Chris Doner - Sybex)

Windows 3.0: Manuel de formation (- Epilog)

Applying Object-Oriented Programming to Graphical User Interface (Zack Urlocker, Glenn Bernsohn - SCOOP West)

Ergonomie.

Systems Application Architecture, Common User Access (- IBM)

L'Interface Utilisateur : Pour une informatique plus conviviale (J.P.Meinadier - Dunod)

Interface homme-ordinateur : Conception et réalisation (j.Coutaz -Dunod)

Cours introductif aux Interfaces Homme-Machine: Notes de cours provisoires (F.Bodart - FUNDP)

Analyse et conception par objets.

Object-Oriented Modeling and Design (J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, W.Lorensen - Prentice Hall)

Object Oriented programming: An Evolutionary Approach (B.J.Cox - Addison Wesley)
Une description orientée objet des objets interactifs abstraits utilisés en Interface Homme-Machine. (I.Provot, J.Vanderdonckt - FUNDP)

Revues spécialisées.

Journal Of Object oriented Programing

Object Magazine

Dr Dobb's

Windows Magazine

Table des matières.

Facultés Universitaires N.D. de la Paix à Namur

Institut d'Informatique

rue Grandgagnage, 21-B 5000 Namur

Spécification et conception d'objets interactifs pour l'environnement Windows 3.0 (Annexe)

Paul BAWIN

Mémoire présenté en vue de l'obtention du titre de Licencié et Maître en Informatique

Promoteur : Monsieur le Professeur François BODART (Institut d'Informatique).

Dans cette annexe, le lecteur trouvera le code des objets interactifs suivants:

- Objet interactif de type "Meter".
- Objet interactif de type "EditLong".
- Objet interactif de type "GraphicButton".
- Objet interactif de type "StateLine".

Le code est organisé suivant les principes de l'environnement *Windows* :

- Le fichier qui inclut le comportement de l'objet interactif d'une part, et les fonctions d'intégration de la classe dans l'environnement d'autre part (*ClassName*.c).
- Le fichier qui inclut la liste des messages publiques (*ClassName*.h). ce fichier devra être intégré dans le code des applications qui utiliseront les services de la classe.
- Le fichier qui inclut les fonctions d'intégration de la classe dans les éditeurs de boîtes de dialogue (*ClassName*Dlg.c).
- Les fichiers des ressources qui contiennent les informations sur la constitution de la boîte de dialogue qui gère la spécialisation d'une instance (*ClassName*.rc et *ClassName*.dlg, Dialog.h).
- Les fichiers qui intègrent des fonctions génériques qui sont utiles pour l'intégration de la classe dans les éditeurs de boîtes de dialogue (Control.c et Control.h).
- Des fichiers qui contiennent quelques fonctions utiles (Util.c et Util.h).
- Le fichier qui permet de définir l'allocation mémoire au niveau du code, de définir les tables des fonctions importées ou exportées, ... (*ClassName*. def).
- Le fichier de construction de gestion de projet (*ClassName*.mak).

1. Objet interactif de type "Meter".

```
/*****************************************************************************
  MODULE: Meter.C

  PURPOSE: Make and manage one meter object

  FUNCTIONS:

  RegisterControlClass() - Register meter control class
  GetValidData()         - Check and update if bad data
  ClearBorder()          - Clear the border of meter control
  DrawBorder()           - Draw th border of meter control
  GetValidBrush()        - Ask to parent to change apparence
  PaintFrameMeter()      - Paint backgroung of meter
  PaintMeter()           - Paint the meter with the given brush
  MeterWndFn()           - process the window messages

*****************************************************************************/

#include <windows.h>

#include "meter.h"


/******* Definition of constants ************************************************/


HANDLE _hInstance       = NULL;
char   _szControlName[] = "Meter";


/******** Definition of control extra bytes ************************************/


#define CBWNDEXTRA          (10)
#define GWL_MAXVALUE         (0)
#define GWL_CURRENTVALUE     (4)
#define GWW_HFONT            (8)

#define OFFSETTOP            (2)
#define OFFSETBOTTOM         (3)


/******** Prototyping **********************************************************/


static short NEAR PASCAL RegisterControlClass (HANDLE hInstance);
LONG FAR PASCAL MeterWndFn (HWND hWnd, WORD wMsg, WORD wParam, LONG lParam);


/********** Window's Dynamic-Link Library Initialization Routines ************/


BOOL FAR PASCAL LibMain (HANDLE hModule, WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdLine)
  {
   if (_hInstance == NULL)
    {
     if (wHeapSize != 0)
       UnlockData(0);
     _hInstance = (RegisterControlClass(hModule) ? hModule : NULL);
     }
   return(_hInstance ? TRUE : FALSE);
  }

VOID FAR PASCAL WEP (int nSystemExit)
```

```
 {
  return;
 }

static short NEAR PASCAL RegisterControlClass (HANDLE hInstance)
 {
  WNDCLASS wc;
  wc.style         = CS_GLOBALCLASS | CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
  wc.lpfnWndProc   = MeterWndFn;
  wc.cbClsExtra    = 0;
  wc.cbWndExtra    = CBWNDEXTRA;
  wc.hInstance     = hInstance;
  wc.hIcon         = NULL;
  wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(LTGRAY_BRUSH);
  wc.lpszMenuName  = NULL;
  wc.lpszClassName = _szControlName;
  return(RegisterClass(&wc));
 }

static VOID NEAR PASCAL GetValidData(WORD FAR *wMax, WORD FAR *wCurrent)
 {
  if (*wMax == 0)
   {
    *wMax = 1;
    *wCurrent = 0;
   }
  else if (*wMax < *wCurrent)
        *wCurrent = *wMax;
 }

static VOID NEAR PASCAL ClearBorder(HDC hDC, RECT rClient, WORD wCurrentValue, WORD wMaxValue)
 {
  MoveTo(hDC, (int)(OFFSETTOP+((rClient.right-OFFSETBOTTOM-OFFSETTOP) * (float)wCurrentValue) / wMaxValue),
              OFFSETTOP);
  LineTo(hDC, (int)(OFFSETTOP+((rClient.right-OFFSETBOTTOM-OFFSETTOP) * (float)wCurrentValue) / wMaxValue),
              rClient.bottom - OFFSETTOP);

 }

static VOID NEAR PASCAL DrawBorder(HDC hDC, RECT rClient, WORD wCurrentValue, WORD wMaxValue)
 {
  POINT          pTab[4];
  HPEN           hPen;

  hPen = SelectObject(hDC, GetStockObject(WHITE_PEN));
  pTab[0].x = OFFSETTOP;
  pTab[0].y = rClient.bottom - OFFSETBOTTOM;
  pTab[1].x = OFFSETTOP;
  pTab[1].y = OFFSETTOP;
  pTab[2].x = (int)(OFFSETTOP+((rClient.right-OFFSETBOTTOM-OFFSETTOP) * (float)wCurrentValue) / wMaxValue);
  pTab[2].y = OFFSETTOP;
  Polyline(hDC, (LPPOINT)&pTab, 3);
  pTab[0].x = (int)(OFFSETTOP+((rClient.right-OFFSETBOTTOM-OFFSETTOP) * (float)wCurrentValue) / wMaxValue);
  pTab[0].y = rClient.bottom - OFFSETBOTTOM;
  pTab[1].x = rClient.right - OFFSETBOTTOM;
  pTab[1].y = rClient.bottom - OFFSETBOTTOM;
  pTab[2].x = rClient.right - OFFSETBOTTOM;
  pTab[2].y = OFFSETTOP;
  Polyline(hDC, (LPPOINT)&pTab, 3);
  hPen = SelectObject(hDC, CreatePen(PS_SOLID, 0, RGB(128, 128, 128)));
  pTab[0].x = OFFSETTOP + (((int)((rClient.right-OFFSETBOTTOM-OFFSETTOP) * (float)wCurrentValue / wMaxValue) > 0)
  pTab[0].y = rClient.bottom - OFFSETBOTTOM;
  pTab[1].x = (int)(OFFSETTOP+((rClient.right-OFFSETBOTTOM-OFFSETTOP) * (float)wCurrentValue) / wMaxValue);
```

```
  pTab[1].y = rClient.bottom - OFFSETBOTTOM;
  pTab[2].x = (int)(OFFSETTOP+((rClient.right-OFFSETBOTTOM-OFFSETTOP) * (float)wCurrentValue) / wMaxValue);
  pTab[2].y = OFFSETTOP;
  pTab[3].x = rClient.right - OFFSETBOTTOM;
  pTab[3].y = OFFSETTOP;
  Polyline(hDC, (LPPOINT)&pTab, 4);
  DeleteObject(SelectObject(hDC, hPen));
 }

static VOID NEAR PASCAL GetValidBrush(HWND hWnd, HDC hDC, HBRUSH FAR *hBrush, DWORD FAR *dwColor)
 {
  HBRUSH   hlBrush;
  LOGBRUSH logbrush;

  hlBrush = (HBRUSH) SendMessage(GetParent(hWnd), WM_CTLCOLOR, hDC,
                                 MAKELONG(hWnd, CTLCOLOR_METER));
  if (!GetObject(hlBrush, sizeof(LOGBRUSH), (LPSTR)&logbrush) ||
      logbrush.lbStyle != BS_SOLID || logbrush.lbColor == RGB(255, 255, 255))
  {
   hlBrush = GetClassWord(hWnd, GCW_HBRBACKGROUND);
   GetObject(hlBrush, sizeof(LOGBRUSH), (LPSTR)&logbrush);
  }
 if (hBrush)
   *hBrush = hlBrush;
 if (dwColor)
  *dwColor = logbrush.lbColor;
 }

static VOID NEAR PASCAL PaintFrameMeter(HWND hWnd, HDC hDC)
 {
  HBRUSH hBrush;
  RECT   rClient;

  GetValidBrush(hWnd, hDC, &hBrush, NULL);
  GetClientRect(hWnd, &rClient);
  FillRect(hDC, &rClient, hBrush);
 }

static VOID NEAR PASCAL PaintMeter(HWND hWnd, HDC hDC)
 {
  HPEN           hPen;
  HFONT          hFont = 0;
  DWORD          dwBkColor;
  WORD           wMaxValue, wCurrentValue;
  WORD           wMaxPrec, wCurrentPrec;
  RECT           rClient;
  char           szPercentage[11];

  wMaxValue = LOWORD(GetWindowLong(hWnd, GWL_MAXVALUE));
  wCurrentValue = LOWORD(GetWindowLong(hWnd, GWL_CURRENTVALUE));
  wMaxPrec = HIWORD(GetWindowLong(hWnd, GWL_MAXVALUE));
  wCurrentPrec = HIWORD(GetWindowLong(hWnd, GWL_CURRENTVALUE));
  GetValidData(&wMaxValue, &wCurrentValue);
  GetValidData(&wMaxPrec, &wCurrentPrec);
  if (GetWindowWord(hWnd, GWW_HFONT))
    hFont = SelectObject(hDC, (HFONT)GetWindowWord(hWnd, GWW_HFONT));
  GetValidBrush(hWnd, hDC, NULL, &dwBkColor);
  GetClientRect(hWnd, &rClient);
  SetTextAlign(hDC, TA_CENTER | TA_TOP);
  SetBkMode(hDC, OPAQUE);
  SetBkColor(hDC, dwBkColor);
  hPen = SelectObject(hDC, CreatePen(PS_SOLID, 0, dwBkColor));
  ClearBorder(hDC, rClient, wCurrentPrec, wMaxPrec);
  DeleteObject(SelectObject(hDC, hPen));
```

```
  wsprintf(szPercentage, "  %d%%  ", (int)((100 * (float)wCurrentValue) / wMaxValue));
  TextOut(hDC, rClient.right / 2,
          (rClient.bottom - HIWORD(GetTextExtent(hDC, "X", 1))) / 2,
          szPercentage, lstrlen(szPercentage));
  DrawBorder(hDC, rClient, wCurrentValue, wMaxValue);
  if (hFont)
    SelectObject(hDC, hFont);
 }

LONG FAR PASCAL MeterWndFn (HWND hWnd, WORD wMsg, WORD wParam, LONG lParam)
 {
  WORD          wPrec;
  LONG          lResult = 0;
  PAINTSTRUCT   ps;
  LOGFONT       logfont;

  switch (wMsg)
    {
     case WM_GETDLGCODE:
          lResult = DLGC_STATIC;
          break;

     case WM_CREATE:
          SetWindowLong(hWnd, GWL_MAXVALUE, MAKELONG(100,100));
          SetWindowLong(hWnd, GWL_CURRENTVALUE, MAKELONG(50,50));
          lResult = TRUE;
          break;

     case WM_SETFONT:
          GetObject((HFONT)wParam, sizeof(logfont), (LPSTR)&logfont);
          SetWindowWord(hWnd, GWW_HFONT, CreateFontIndirect(&logfont));
          if (lParam)
           {
            InvalidateRect(hWnd, NULL, FALSE);
            UpdateWindow(hWnd);
           }
          lResult = TRUE;
          break;

     case WM_ERASEBKGND:
          PaintFrameMeter(hWnd, (HDC)wParam);
          lResult = TRUE;
          break;

     case WM_PAINT:
          BeginPaint(hWnd, &ps);
          PaintMeter(hWnd, ps.hdc);
          EndPaint(hWnd, &ps);
          lResult = TRUE;
          break;

     case MM_SETMAXVALUE:
          wPrec = LOWORD(GetWindowLong(hWnd, GWL_MAXVALUE));
          SetWindowLong(hWnd, GWL_MAXVALUE, MAKELONG(wParam,wPrec));
          InvalidateRect(hWnd, NULL, FALSE);
          UpdateWindow(hWnd);
          lResult = TRUE;
          break;

     case MM_GETMAXVALUE:
          lResult = (LONG)LOWORD(GetWindowLong(hWnd, GWL_MAXVALUE));
          break;

     case MM_SETCURRENTVALUE:
```

```
            wPrec = LOWORD(GetWindowLong(hWnd, GWL_CURRENTVALUE));
            SetWindowLong(hWnd, GWL_CURRENTVALUE, MAKELONG(wParam, wPrec));
            InvalidateRect(hWnd, NULL, FALSE);
            UpdateWindow(hWnd);
            lResult = TRUE;
            break;

        case MM_GETCURRENTVALUE:
            lResult = (LONG)LOWORD(GetWindowLong(hWnd, GWL_CURRENTVALUE));
            break;

        default:
            lResult = DefWindowProc(hWnd, wMsg, wParam, lParam);
            break;
      }
    return(lResult);
}
```

```
/*****************************************************************************
  MODULE: Control.C

  PURPOSE: Contain the functions (generic) to interface object with dlg box
           editor.

  FUNCTIONS:

  ControlInfo()    - Allocate memory and stock info into structure
  AddControlType() - Add into dlg editor the control type of object
  ShowStyleDlg()   - Display in dlg editor the dialog box to select
                     style of object
  CtlStyleLock()   - Lock memory block which contain styles
  CtlStyleUnlock() - Unlock memory block which contain styles
  GetIdString()    - Call spcifique functions to convert string to an ID
  SetIdValue()     -  "      "          "

*****************************************************************************/

#include <windows.h>
#include <custcntl.h>

#include "control.h"


/********** Definition of structures and constants ***************************/


static char _szCtlProp[] = "CtlDlgStyleData";

typedef struct {
                GLOBALHANDLE hCtlStyle;
                LPFNSTRTOID  lpfnStrToId;
                LPFNIDTOSTR  lpfnIdToStr;
               } CTLSTYLEDLG, FAR *LPCTLSTYLEDLG, NEAR *NPCTLSTYLEDLG;



GLOBALHANDLE FAR PASCAL ControlInfo (WORD wVersion, LPSTR szClass, LPSTR szTitle)
 {
  GLOBALHANDLE hMem = NULL;
  LPCTLINFO    lpCtlInfo;

  hMem = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, (DWORD) sizeof(CTLINFO));
  if (hMem == NULL)
    return(hMem);
  lpCtlInfo = (LPCTLINFO) GlobalLock(hMem);
  lpCtlInfo->wVersion  = wVersion;

  lpCtlInfo->wCtlTypes = 0;
  lstrcpy(lpCtlInfo->szClass, szClass);
  lstrcpy(lpCtlInfo->szTitle, szTitle);
  GlobalUnlock(hMem);
  return(hMem);
 }

BOOL FAR PASCAL AddControlType (GLOBALHANDLE hMem, WORD wType, WORD wWidth,
                                WORD wHeight, DWORD dwStyle, LPSTR szDescr)
 {
  LPCTLINFO lpCtlInfo; WORD wNumTypes;
  lpCtlInfo = (LPCTLINFO) GlobalLock(hMem);
  wNumTypes = lpCtlInfo->wCtlTypes;
  if (wNumTypes == CTLTYPES)
```

```
    {
     GlobalUnlock(hMem);
     return(FALSE);
    }
   lpCtlInfo->Type[wNumTypes].wType   = wType;
   lpCtlInfo->Type[wNumTypes].wWidth  = wWidth;
   lpCtlInfo->Type[wNumTypes].wHeight = wHeight;
   lpCtlInfo->Type[wNumTypes].dwStyle = dwStyle;
   lstrcpy(lpCtlInfo->Type[wNumTypes].szDescr, szDescr);
   lpCtlInfo->wCtlTypes++;
   GlobalUnlock(hMem);
   return(TRUE);
  }

int FAR PASCAL ShowStyleDlg (HANDLE hInstance, LPSTR szTemplate,
                             HWND hWndParent, FARPROC fpDlgProc, LONG lParam,
                             GLOBALHANDLE hCtlStyle, LPFNSTRTOID lpfnStrToId,
                             LPFNIDTOSTR lpfnIdToStr)
  {
   LOCALHANDLE hCtlStyleDlg;
   NPCTLSTYLEDLG npCtlStyleDlg;
   int x;

   hCtlStyleDlg = LocalAlloc(LMEM_MOVEABLE | LMEM_ZEROINIT, sizeof(CTLSTYLEDLG));
   if (hCtlStyleDlg == NULL)
     return(FALSE);
   npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
   npCtlStyleDlg->hCtlStyle = hCtlStyle;
   npCtlStyleDlg->lpfnStrToId = lpfnStrToId;
   npCtlStyleDlg->lpfnIdToStr = lpfnIdToStr;
   LocalUnlock(hCtlStyleDlg);
   SetProp(hWndParent, _szCtlProp, hCtlStyleDlg);
   x = DialogBoxParam(hInstance, szTemplate, hWndParent, fpDlgProc, lParam);
   RemoveProp(hWndParent, _szCtlProp);
   LocalFree(hCtlStyleDlg);
   return(x == IDOK);
  }

LPCTLSTYLE FAR PASCAL CtlStyleLock (HWND hDlg)
  {
   LOCALHANDLE   hCtlStyleDlg;
   NPCTLSTYLEDLG npCtlStyleDlg;
   LPCTLSTYLE    lpCtlStyle = NULL;

   hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
   if (hCtlStyleDlg == NULL)
     return(lpCtlStyle);
   npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
   lpCtlStyle = (LPCTLSTYLE) GlobalLock(npCtlStyleDlg->hCtlStyle);
   LocalUnlock(hCtlStyleDlg);
   return(lpCtlStyle);
  }

BOOL FAR PASCAL CtlStyleUnlock (HWND hDlg)
  {
   LOCALHANDLE   hCtlStyleDlg;
   NPCTLSTYLEDLG npCtlStyleDlg;
   BOOL          fOk = FALSE;

   hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
   if (hCtlStyleDlg == NULL)
     return(fOk);
   npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
   fOk = GlobalUnlock(npCtlStyleDlg->hCtlStyle);
```

```
  LocalUnlock(hCtlStyleDlg);
  return(fOk);
 }

WORD FAR PASCAL GetIdString (HWND hDlg, LPSTR szId, WORD wIdMaxLen)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle;
  WORD          wIdLen;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(0);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  lpCtlStyle = (LPCTLSTYLE) GlobalLock(npCtlStyleDlg->hCtlStyle);
  wIdLen = (*npCtlStyleDlg->lpfnIdToStr)(lpCtlStyle->wId, szId, wIdMaxLen);
  GlobalUnlock(npCtlStyleDlg->hCtlStyle);
  LocalUnlock(hCtlStyleDlg);
  return(wIdLen);
 }

DWORD FAR PASCAL SetIdValue (HWND hDlg, LPSTR szId)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle;
  DWORD         dwResult = 0;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(dwResult);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  dwResult = (*npCtlStyleDlg->lpfnStrToId)(szId);
  LocalUnlock(hCtlStyleDlg);
  if (LOWORD(dwResult) == 0)
    return(dwResult);
  lpCtlStyle = CtlStyleLock(hDlg);
  lpCtlStyle->wId = HIWORD(dwResult);
  CtlStyleUnlock(hDlg);
  return(dwResult);
 }
```

```
/*****************************************************************************
  MODULE: MeterDlg.C

  PURPOSE: This file contain the functions to integrate the Long
           object into dialog boxes tools generator.

  FUNCTIONS:

  MeterInfo()  - Return to Windows the name of class,
                 the version and the title of object
  MeterStyle() - Is called by dialog box tools for used in this
                 the dialog box of specialization
  MeterDlgFn() - Function to process the dlg messages
  MeterFlags() - Build string to integrate in resources

*****************************************************************************/

#include <windows.h>
#include <custcntl.h>

#include "control.h"

#include "meter.h"
#include "dialog.h"


/********* Definition of constants ***************************************/


extern HANDLE _hInstance;
extern char _szControlName[];


/********* Prototyping ***************************************************/

BOOL FAR PASCAL MeterDlgFn (HWND hDlg, WORD wMessage, WORD wParam, LONG lParam);




GLOBALHANDLE FAR PASCAL MeterInfo (void)
{
   GLOBALHANDLE hCtlInfo = NULL;

   hCtlInfo = ControlInfo(0x0100, _szControlName, _szControlName);
   if (hCtlInfo == NULL) return(hCtlInfo);
   AddControlType(hCtlInfo, 0, 40, 12, WS_BORDER | WS_CHILD, _szControlName);
   return(hCtlInfo);

}

BOOL FAR PASCAL MeterStyle (HWND hWnd, GLOBALHANDLE hCtlStyle,
                        LPFNSTRTOID lpfnStrToId, LPFNIDTOSTR lpfnIdToStr)
{
   return(ShowStyleDlg(_hInstance, "StyleDlg", hWnd, (FARPROC) MeterDlgFn,
                       0, hCtlStyle, lpfnStrToId, lpfnIdToStr));
}

BOOL FAR PASCAL MeterDlgFn (HWND hDlg, WORD wMsg, WORD wParam, LONG lParam) {
   BOOL fResult = TRUE;
   char szId[20];
   DWORD dwResult;

   switch (wMsg) {
```

```
        case WM_INITDIALOG:
           GetIdString(hDlg, szId, sizeof(szId));
           SetDlgItemText(hDlg, ID_VALUE, szId);
           break;

        case WM_COMMAND:
           switch (wParam) {
              case IDOK:
                   GetDlgItemText(hDlg, ID_VALUE, szId, sizeof(szId));
                   dwResult = SetIdValue(hDlg, szId);
                   if (LOWORD(dwResult) == 0) break;

              case IDCANCEL:
                   EndDialog(hDlg, wParam);
                   break;

              case ID_VALUE:
                   if (HIWORD(lParam) == EN_CHANGE)
                     EnableWindow(GetDlgItem(hDlg, IDOK),
                           SendMessage(LOWORD(lParam), WM_GETTEXTLENGTH, 0, 0L)
                           ? TRUE : FALSE);
                   break;

              default: fResult = FALSE;
                       break;
           }
           break;

        default: fResult = FALSE;
                 break;
     }
     return(fResult);
}

WORD FAR PASCAL MeterFlags (DWORD dwFlags, LPSTR szString, WORD wMaxString) {
     WORD x;
     *szString = 0;
     x = lstrlen(szString);
     if (x > 0) { x -= sizeof(" | ") - 1; *(szString + x) = 0; }
     return(x);
}
```

```
/*****************************************************************************
File name: Control.H
*****************************************************************************/

GLOBALHANDLE FAR PASCAL ControlInfo (WORD wVersion, LPSTR szClass, LPSTR szTitle);
BOOL FAR PASCAL AddControlType (GLOBALHANDLE hMem, WORD wType, WORD wWidth,
                                WORD wHeight, DWORD dwStyle, LPSTR szDescr);
int FAR PASCAL ShowStyleDlg (HANDLE hInstance, LPSTR szTemplate,
                             HWND hWndParent, FARPROC fpDlgProc, LONG lParam,
                             GLOBALHANDLE hCtlStyle, LPFNSTRTOID lpfnStrToId,
                             LPFNIDTOSTR lpfnIdToStr);
LPCTLSTYLE FAR PASCAL CtlStyleLock (HWND hDlg);
BOOL FAR PASCAL CtlStyleUnlock (HWND hDlg);
WORD FAR PASCAL GetIdString (HWND hDlg, LPSTR szId, WORD wIdMaxLen);
DWORD FAR PASCAL SetIdValue (HWND hDlg, LPSTR szId);
```

```
#define ID_VALUE        100

#define IDOK            1
#define IDCANCEL        2
```

```
/*****************************************************************************
File name: Meter.H
*****************************************************************************/

#define CTLCOLOR_METER      (20)

// Meter control's class-specific window styles.


// Meter control's class-specific window messages.
#define MM_SETMAXVALUE          (WM_USER + 0)
#define MM_GETMAXVALUE          (WM_USER + 1)
#define MM_SETCURRENTVALUE      (WM_USER + 2)
#define MM_GETCURRENTVALUE      (WM_USER + 3)

// Meter control's notification codes to send to parent.
```

```
STYLEDLG DIALOG LOADONCALL MOVEABLE DISCARDABLE 83, 73, 156, 36
CAPTION "Meter Style..."
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "&ID Value:", -1, "static", SS_LEFT | WS_CHILD, 4, 4, 32, 12
    CONTROL "", ID_VALUE, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 36, 4, 116, 12
    CONTROL "&Ok", IDOK, "button", BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD, 36, 20, 32, 12
    CONTROL "&Cancel", IDCANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 88, 20, 32, 12
END
```

```
; Module name: Meter.DEF
; Programmer : Jeffrey M. Richter

LIBRARY     METER
DESCRIPTION 'Meter Custom Control Library'
EXETYPE     WINDOWS
STUB        'WinStub.Exe'

; CODE        PRELOAD FIXED NONDISCARDABLE
; DATA        MOVEABLE PRELOAD SINGLE

CODE        MOVEABLE DISCARDABLE SHARED PRELOAD
DATA        MOVEABLE SINGLE PRELOAD

HEAPSIZE    1024

EXPORTS
   WEP            @1    RESIDENTNAME
   MeterInfo      @2
   MeterStyle     @3
   MeterFlags     @4
   MeterWndFn     @5
   MeterDlgFn     @6
```

```
/*****************************************************************************
File name: Meter.RC
*****************************************************************************/

#include <windows.h>

#include "meter.h"
#include "dialog.h"
#include "meter.dlg"
```

```
#*****************************************************************************
#File name: MAKEFILE
#*****************************************************************************

PROG  = Meter
MODEL = S
CFLAGS = -A$(MODEL)w -D_WINDOWS -D_WINDLL -Gcsw2 -W4 -Zlepid -Od
LFLAGS = /NOE/BA/A:16/M/CO/LI/F
LIBS = $(MODEL)nocrtd + $(MODEL)dllcew + libw

M1 = $(PROG).obj $(PROG)Dlg.obj Control.obj

ICONS   =
BITMAPS =
CURSORS =
RESOURCES = $(ICONS) $(BITMAPS) $(CURSORS)

.SUFFIXES: .rc

.rc.res:
    rc -r $*.rc

$(PROG).DLL: $(M1) $(PROG).Def $(PROG).Res
     link $(LFLAGS) @<<$(PROG).Lnk
C:\WINDEV\LIB\LibEntry.obj $(M1)
$(PROG).DLL, $(PROG), $(LIBS), $(PROG)
<<
     rc -Fe$(PROG).DLL $(PROG).Res
     copy $(PROG).dll c:\win3

$(PROG).obj:    $*.c $*.h
$(PROG)Dlg.obj: $*.c $(PROG).h  dialog.h
Control.obj:    $*.c $*.h

$(PROG).res:   $*.rc $*.h $*.dlg dialog.h $(RESOURCES)
```

2. Objet interactif de type "EditLong"

```
/*****************************************************************************
  MODULE: Long.C

  PURPOSE: Make and manage an edit box for acquisition of a long integer only

  FUNCTIONS:

  RegisterControlClass() - Register Long edit box control class
  LongAddress()          - Initialise values with a constant
  CheckOutOfRange()      - Check if the value is in range defined
                           by the user
  StrToLong()            - Translate the string to a long integer
  PosCharInString()      - Check if character and position is correct
  CheckClipboard()       - Check if data is valid in clipboard
  StartLong()            - Check if prefix of data is valid
  GetCurrentValue()      - Return the current value if it is valid
  SetCurrentValue()      - Initialize with with value given
  LongWndFn()            - Process window message for th Long control


*****************************************************************************/

#include <windows.h>
#include <math.h>
#include <stdlib.h>

#include "util.h"
#include "supercls.h"
#include "long.h"

/********** Definition of constants ******************************************/

HANDLE  _hInstance = NULL;
char    _szControlName[] = "Long";


/********** Definition of control extra bytes ********************************/


#define CBWNDEXTRA     (8)
#define GWL_MINVALUE   (0)
#define GWL_MAXVALUE   (4)
#define CBCLSEXTRA     (0)

#define GetStyle       GetWindowLong(hWnd, GWL_STYLE)
#define SetStyle(x)    SetWindowLong(hWnd, GWL_STYLE, x)

#define IsMinNeg       ((LONG)GetSCWindowLong(hWnd, GWL_MINVALUE) < 0 ? TRUE : FALSE)
#define IsMaxPos       ((LONG)GetSCWindowLong(hWnd, GWL_MAXVALUE) >= 0 ? TRUE : FALSE)


/********* Prototyping *******************************************************/


static short NEAR PASCAL RegisterControlClass (HANDLE hInstance);
LONG FAR PASCAL LongWndFn (HWND hWnd, WORD wMsg, WORD wParam, LONG lParam);


/********** Window's Dynamic-Link Library Initialization Routines ************/

BOOL FAR PASCAL LibMain (HANDLE hModule, WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdLine)
  {
   if (_hInstance == NULL)
    {
```

```
    if (wHeapSize != 0)
      UnlockData(0);
    _hInstance = (RegisterControlClass(hModule) ? hModule : NULL);
    }
   return(_hInstance ? TRUE : FALSE);
  }

VOID FAR PASCAL WEP (int nSystemExit)
 {
  return;
 }

static short NEAR PASCAL RegisterControlClass (HANDLE hInstance)
  {
   WNDCLASS wc;

   GetClassInfo(NULL, "EDIT", &wc);

   wc.style         = CS_GLOBALCLASS | CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
   wc.hInstance     = hInstance;
   wc.lpszMenuName  = NULL;
   wc.lpszClassName = _szControlName;
   return (RegisterSuperClass(&wc, LongWndFn, CBCLSEXTRA, CBWNDEXTRA));
  }

LONG FAR PASCAL LongAddress(LONG lValue)
 {
  static LONG lStaticValue;
  lStaticValue = lValue;
  return (LONG)(LPLONG)&lStaticValue;
 }

static BOOL NEAR PASCAL CheckOutOfRange(HWND hWnd, LONG lValue)
 {
  if ((LONG)GetSCWindowLong(hWnd, GWL_MINVALUE) <= lValue &&
      (LONG)GetSCWindowLong(hWnd, GWL_MAXVALUE) >= lValue)
    return (TRUE);
  return (FALSE);
 }

static BOOL NEAR PASCAL StrToLong(LPSTR szString, LPLONG lpValue)
 {
  int   nSize;
  int   nIndice = 0;
  DWORD dwValue = 0;
  BOOL  bResult = FALSE;

  nSize = lstrlen(szString) - 1;
  while (nSize >= 0 && szString[nSize] == ' ') nSize--;
  while (nSize >= 0 && lCharInString(szString[nSize], "0123456789") && (nIndice < 9 || (nIndice == 9 && szString[
   {
    dwValue += ((szString[nSize] - 48) * (DWORD)pow(10.0, (double)nIndice));
    nIndice++;
    nSize--;
   }
  if ((bResult = dwValue <= MAXLONG ? TRUE : FALSE) && lpValue)
    *lpValue = (LONG)dwValue;
  while (nSize >= 0 && szString[nSize] == ' ') nSize--;
  if ((nSize >= 0) && ((szString[nSize] == '-') || (szString[nSize] == '+')))
   {
    if (lpValue && (szString[nSize] == '-'))
      *lpValue = -(*lpValue);
    nSize--;
   }
```

```
  while (nSize >= 0 && szString[nSize] == ' ') nSize--;
  if (nSize >= 0)
    bResult = FALSE;
  return bResult;
 }

static BOOL NEAR PASCAL PosCharInString(HWND hWnd, char car, BOOL bNeg, BOOL bStrict, LPLONG lplCode)
  {
   LONG   lPos;
   WORD   wSize, wStart;
   HANDLE hStr;
   LPSTR  lpStr;
   BOOL   bResult = FALSE;
   char   szStr[20];

   *lplCode = LGN_BADCHAR;
   if (car == VK_BACK || car == VK_DELETE ||
       car == '-' && (IsMinNeg && bStrict || !bStrict) ||
       car == '+' && (IsMaxPos && bStrict || !bStrict) ||
       lCharInString(car, (LPSTR)"0123456789 ")
      )
    {
     wSize = (WORD)GetWindowTextLength(hWnd) + 2; // +1
     if ((hStr = GlobalAlloc(GHND, wSize * sizeof(char))) == NULL)
       return (FALSE);
     lpStr = (char FAR *)GlobalLock(hStr);
     GetWindowText(hWnd, lpStr, wSize);
     lstrcpy((LPSTR)szStr, lpStr);
     lPos = SendMessage(hWnd, EM_GETSEL, 0, 0L);
     wStart = min(LOWORD(lPos), HIWORD(lPos));
     lDeleteCharInString(lpStr, wStart, max(LOWORD(lPos),HIWORD(lPos)) - wStart);
     lstrcpy((LPSTR)szStr, lpStr);
     *lplCode = LGN_BADPOSCHAR;
     switch (car)
      {
       case VK_SPACE:
            if (wStart == 0 || lpStr[wStart-1] == ' ' || lpStr[wStart-1] == '-' || lpStr[wStart-1] == '+')
              bResult = TRUE;
            break;

       case 43:
       case 45:
            if ( !lCharInString(car, lpStr) &&
                (wStart == 0 || lpStr[wStart-1] == ' '))
              bResult = TRUE;
            break;

       case VK_DELETE:
            wStart++;
       case VK_BACK:
            if (bStrict && bNeg && lCharInString('-', lpStr) &&
                (WORD)lCharInString('-', lpStr) != wStart)
              bResult = TRUE;
            if (bStrict && bNeg && wStart == (WORD)lstrlen(lpStr) &&
                (WORD)lCharInString('-', lpStr) == wStart)
              bResult = TRUE;
            if (!bStrict || bStrict && !bNeg)
              bResult = TRUE;
            if (LOWORD(lPos) != HIWORD(lPos) && lCharInString('-', lpStr))
              bResult = TRUE;
            if (LOWORD(lPos) != HIWORD(lPos) && lAllCharInString(' ', lpStr))
              bResult = TRUE;
            break;
```

```
        default:
             if (bNeg && lCharInString('-', lpStr) && bStrict ||
                 bNeg && !bStrict || !bNeg)
               bResult = TRUE;
             break;
        }
       if (bResult && car != VK_BACK && car != VK_DELETE)
        {
         lInsertCharInString(lpStr, wStart, car);
         *lplCode = (bResult = StrToLong(lpStr, NULL)) ? NULL : LGN_OUTOFRANGE;
        }
       GlobalUnlock(hStr);
       GlobalFree(hStr);
      }
     return (bResult);
    }

static BOOL NEAR PASCAL CheckClipboard(HWND hWnd, BOOL bNeg, BOOL bStrict, LPLONG lplCode)
    {
     LONG          lPos;
     LONG          lValue;
     WORD          wSize, wStart, wNumber, wCurrent;
     HANDLE        hData = NULL, hStr = NULL;
     LPSTR         lpStr, lpData;
     BOOL          bResult = FALSE;

     *lplCode = LGN_ERROR;
     if (!OpenClipboard(hWnd))
       return bResult;
     if (!(hData = GetClipboardData(CF_TEXT)))
      {
       *lplCode = LGN_BADFORMAT;
       CloseClipboard();
       return bResult;
      }
     lpData = GlobalLock(hData);
     lPos = SendMessage(hWnd, EM_GETSEL, 0, 0L);
     wStart = min(LOWORD(lPos), HIWORD(lPos));
     wNumber = max(LOWORD(lPos),HIWORD(lPos)) - wStart;
     wSize = wNumber > (WORD)lstrlen(lpData) ? 0 : (WORD)lstrlen(lpData) - wNumber;
     wSize += (WORD)GetWindowTextLength(hWnd) + 1;
     *lplCode = LGN_ERROR;
     if ((hStr = GlobalAlloc(GHND, wSize * sizeof(char))))
      {
       lpStr = (char FAR *)GlobalLock(hStr);
       GetWindowText(hWnd, lpStr, wSize);
       lDeleteCharInString(lpStr, wStart, wNumber);
       wSize = lstrlen(lpData);
       for (wCurrent = 0; wCurrent < wSize; wCurrent++)
         lInsertCharInString(lpStr, wStart++, lpData[wCurrent]);
       *lplCode = (bResult = StrToLong(lpStr, &lValue)) ? NULL : LGN_BADFORMAT;
       if (!bNeg && bStrict && lCharInString('-', lpStr) ||
           bNeg && bStrict && !lCharInString('-', lpStr))
        {
         bResult = FALSE;
         *lplCode = LGN_OUTOFRANGE;
        }
       GlobalUnlock(hStr);
       GlobalFree(hStr);
      }
     GlobalUnlock(hData);
     CloseClipboard();
     return (bResult);
    }
```

```
static BOOL NEAR PASCAL StartLong(LPSTR lpStr)
 {
  int nSize = lstrlen(lpStr);
  int nCur;

  for (nCur = 0; nCur < nSize; nCur++)
   if (lpStr[nCur] != ' ' && lpStr[nCur] != '+' && lpStr[nCur] != '-')
     return FALSE;
  return TRUE;
 }

static LONG NEAR PASCAL GetCurrentValue(HWND hWnd, WORD wParam, LPLONG lParam)
 {
  HANDLE hStr;
  LPSTR  lpStr;
  LONG   lResult;
  WORD   wSize;

  if ((lResult = wParam == sizeof(LONG) ? LG_OK : LG_ERR) == LG_OK)
    if ((lResult = (wSize = (WORD)SendMessage(hWnd, WM_GETTEXTLENGTH, 0, 0L)) == 0 ? LG_NULL : LG_OK) == LG_OK)
      if ((lResult = (hStr = GlobalAlloc(GHND, ++wSize * sizeof(char))) == NULL ? LG_ERR : LG_OK) == LG_OK)
       {
        lpStr = (LPSTR)GlobalLock(hStr);
        if ((lResult = SendMessage(hWnd, WM_GETTEXT, wSize, (LONG)lpStr) == LB_ERR ? LG_ERR : LG_OK) == LG_OK)
           if ((lResult = !StartLong(lpStr) ? LG_OK : LG_NULL) == LG_OK)
             if ((lResult = StrToLong(lpStr, lParam) ? LG_OK : LG_BADFORMAT) == LG_OK)
              lResult = CheckOutOfRange(hWnd, *lParam) ? LG_OK : LG_OUTOFRANGE;
        GlobalUnlock(hStr);
        GlobalFree(hStr);
       }
  return lResult;
 }

static LONG NEAR PASCAL SetCurrentValue(HWND hWnd, LPLONG lParam)
 {
  char strValue[12];
  LONG lResult;

  if ((lResult = lParam != NULL ? LG_OK : LG_NULL) == LG_OK)
   {
    if ((lResult = CheckOutOfRange(hWnd, *lParam) ? LG_OK : LG_OUTOFRANGE) == LG_OK)
      wsprintf((LPSTR)strValue, "%ld", *lParam);
   }
  else
    lstrcpy((LPSTR)strValue, (LPSTR)"");
  if (lResult == LG_NULL || lResult == LG_OK ||
      lResult == LG_OUTOFRANGE && !(GetStyle & LGS_STRICT))
    SendMessage(hWnd, WM_SETTEXT, 0, (LONG)(LPSTR)strValue);
  return lResult;
 }

LONG FAR PASCAL LongWndFn (HWND hWnd, WORD wMsg, WORD wParam, LONG lParam)
 {
  LONG         lResult = 0;
  LONG         lValue  = 0;
  BOOL         bCallBase = TRUE;

  switch (wMsg)
     {
      case WM_NCCREATE:
           SetSuperClassInfo(hWnd);
           break;
```

```
case WM_CREATE:
     SetSCWindowLong(hWnd, GWL_MINVALUE, MINLONG);
     SetSCWindowLong(hWnd, GWL_MAXVALUE, MAXLONG);
     break;

case WM_CHAR:
     if (!PosCharInString(hWnd, (char)wParam, IsMinNeg && !IsMaxPos, (BOOL)(GetStyle & LGS_STRICT), &lValue
      {
       bCallBase = FALSE;
       if (GetStyle & LGS_BEEP)
         MessageBeep(0);
       Notification(hWnd, (WORD)lValue);
      }
     break;

case WM_KEYDOWN:
     if (wParam == VK_INSERT && GetKeyState(VK_SHIFT) & 0x8000 &&
         !CheckClipboard(hWnd, IsMinNeg && !IsMaxPos, (BOOL)(GetStyle & LGS_STRICT), &lValue))
      {
       if (GetStyle & LGS_BEEP)
         MessageBeep(0);
       bCallBase = FALSE;
       Notification(hWnd, (WORD)lValue);
      }
     if (wParam == VK_DELETE &&
         !PosCharInString(hWnd, (char)wParam, IsMinNeg && !IsMaxPos, (BOOL)(GetStyle & LGS_STRICT), &lValue
      {
       bCallBase = FALSE;
       if (GetStyle & LGS_BEEP)
         MessageBeep(0);
       Notification(hWnd, (WORD)lValue);
      }
     break;

case WM_PASTE:
     if (!CheckClipboard(hWnd, IsMinNeg && !IsMaxPos, (BOOL)(GetStyle & LGS_STRICT), &lValue))
      {
       if (GetStyle & LGS_BEEP)
         MessageBeep(0);
       bCallBase = FALSE;
       Notification(hWnd, (WORD)lValue);
      }
     break;

case WM_SETTEXT:
     if ((lResult = lstrlen((LPSTR)lParam) ? LG_OK : LG_NULL) == LG_OK)
      {
       if ((lResult = !StrToLong((LPSTR)lParam, &lValue) ? LG_BADFORMAT : LG_OK) == LG_BADFORMAT)
         bCallBase = FALSE;
       else if ((lResult = !CheckOutOfRange(hWnd, lValue) ? LG_OUTOFRANGE : LG_OK) == LG_OUTOFRANGE)
              bCallBase = FALSE;
      }
     break;

case WM_KILLFOCUS:
     if (SendMessage(hWnd, LGM_CHECKCURRENTVALUE, 0, 0L) == LG_NULL)
       SetWindowText(hWnd, (LPSTR)"");
     break;

case LGM_SETMINVALUE:
     SetSCWindowLong(hWnd, GWL_MINVALUE, lParam != NULL ? *(LPLONG)lParam : MINLONG);
     lResult = GetSCWindowLong(hWnd, GWL_MINVALUE);
     bCallBase = FALSE;
     break;
```

```
    case LGM_GETMINVALUE:
         if ((lResult = wParam == sizeof(LONG) ? LG_OK : LG_ERR) == LG_OK)
           *(LPLONG)lParam = GetSCWindowLong(hWnd, GWL_MINVALUE);
         bCallBase = FALSE;
         break;

    case LGM_SETMAXVALUE:
         SetSCWindowLong(hWnd, GWL_MAXVALUE, lParam != NULL ? *(LPLONG)lParam : MAXLONG);
         lResult = GetSCWindowLong(hWnd, GWL_MAXVALUE);
         bCallBase = FALSE;
         break;

    case LGM_GETMAXVALUE:
         if ((lResult = wParam == sizeof(LONG) ? LG_OK : LG_ERR) == LG_OK)
           *(LPLONG)lParam = GetSCWindowLong(hWnd, GWL_MAXVALUE);
         bCallBase = FALSE;
         break;

    case LGM_SETCURRENTVALUE:
         lResult = SetCurrentValue(hWnd, (LPLONG)lParam);
         bCallBase = FALSE;
         break;

    case LGM_GETCURRENTVALUE:
         lResult = GetCurrentValue(hWnd, wParam, (LPLONG)lParam);
         bCallBase = FALSE;
         break;

    case LGM_CHECKCURRENTVALUE:
         lResult = GetCurrentValue(hWnd, sizeof(LONG), &lValue);
         bCallBase = FALSE;
         break;

    case LGM_SETALLVALUES:
         if (wParam >= sizeof(LONG))
           SendMessage(hWnd, LGM_SETMINVALUE, 0, lParam);
         if (wParam >= 2*sizeof(LONG))
           SendMessage(hWnd, LGM_SETMAXVALUE, 0, (LONG)((LPLONG)lParam+1));
         if (wParam == 3*sizeof(LONG))
           SendMessage(hWnd, LGM_SETCURRENTVALUE, 0, (LONG)((LPLONG)lParam+2));
         bCallBase = FALSE;
         break;

    default:
         break;
   }
 if (bCallBase)
   lResult = CallWindowProc((FARPROC)GetBCWndProc(hWnd), hWnd, wMsg, wParam, lParam);
 return(lResult);
}
```

```
/*****************************************************************************
  MODULE: SuperCls.C

  PURPOSE: Functions to manage inheritence mechanism

  FUNCTIONS:

  RegisterSuperClass() - Register the base class with info
                         on super class
  SetSuperClassInfo()  - Stores information about superclassing
  GetBCWndProc()       - Returns the function of the inherited class
  CalcClassIndex()     - Defines indexes for sup classes
  CalcWindowIndex()    - Defines indexes for this class
  SetSCClassWord()     - Initializes indexes of super class
  GetSCClassWord()     - Retreives indexes of super class
  SetSCClassLong()     - Set info of super class
  GetSCClassLong()     - Retreives info of super class
  SetSCWindowWord()    - Set info this class
  GetSCWindowWord()    - Retreives info of this class
  SetSCWindowLong()    - Set private data of class
  GetSCWindowLong()    - Get private data of class

*****************************************************************************/

#include <windows.h>

#include "supercls.h"

/********* Definition of constants ***************************************/

#define CBCLSPROCINDEX (sizeof(FARPROC))
#define CBCLSECLSINDEX (sizeof(FARPROC) + sizeof(int))
#define CBCLSEWNDINDEX (sizeof(FARPROC) + sizeof(int) + sizeof(int))
#define CBCLSEADDITION (sizeof(FARPROC) + sizeof(int) + sizeof(int))

/********* Definition of globales variables ******************************/

static int     _cbClsExtraSuper;
static int     _cbWndExtraSuper;
static LONG    (FAR PASCAL *_lpfnWndProcSuper)();



BOOL FAR PASCAL RegisterSuperClass(LPWNDCLASS lpWC, LONG (FAR PASCAL *lpfnSCWndProc)(),
                                   int cbClsAdditional, int cbWndAdditional)
 {
  HWND hWnd;

  _lpfnWndProcSuper = lpWC->lpfnWndProc;
  _cbClsExtraSuper = lpWC->cbClsExtra;
  _cbWndExtraSuper = lpWC->cbWndExtra;

  lpWC->cbClsExtra += cbClsAdditional + CBCLSEADDITION;
  lpWC->cbWndExtra += cbWndAdditional;
  lpWC->lpfnWndProc = lpfnSCWndProc;
  if (!RegisterClass(lpWC))
    return FALSE;
  if ((hWnd = CreateWindow(lpWC->lpszClassName, "", 0, 0, 0, 0, 0, NULL, NULL,
                           lpWC->hInstance, 0)) == NULL)
   {
    UnregisterClass(lpWC->lpszClassName, lpWC->hInstance);
    return FALSE;
   }
  DestroyWindow(hWnd);
```

```
  return TRUE;
 }

void FAR PASCAL SetSuperClassInfo(HWND hWnd)
 {
  WORD cbClsTotal = GetClassWord(hWnd, GCW_CBCLSEXTRA);
  DWORD dwBCWndProc = GetClassLong(hWnd, cbClsTotal - CBCLSPROCINDEX);
  if (dwBCWndProc != NULL)
    return;
  SetClassLong(hWnd, cbClsTotal - CBCLSPROCINDEX, (LONG)_lpfnWndProcSuper);
  SetClassWord(hWnd, cbClsTotal - CBCLSECLSINDEX, _cbClsExtraSuper);
  SetClassWord(hWnd, cbClsTotal - CBCLSEWNDINDEX, _cbWndExtraSuper);
 }

DWORD FAR PASCAL GetBCWndProc(HWND hWnd)
 {
  int nIndex = GetClassWord(hWnd, GCW_CBCLSEXTRA) - CBCLSPROCINDEX;
  DWORD dwBCWndProc = GetClassLong(hWnd, nIndex);
  if (dwBCWndProc != NULL)
    return (dwBCWndProc);
  return ((DWORD)_lpfnWndProcSuper);
 }

static int NEAR PASCAL CalcClassIndex(HWND hWnd, int nIndex)
 {
  int cbBCClsIndex, cbBCClsExtra;

  if (nIndex < 0)
    return (nIndex);
  cbBCClsIndex = GetClassWord(hWnd, GCW_CBCLSEXTRA) - CBCLSECLSINDEX;
  cbBCClsExtra = GetClassWord(hWnd, cbBCClsIndex);
  return (cbBCClsExtra + nIndex);
 }

static int NEAR PASCAL CalcWindowIndex(HWND hWnd, int nIndex)
 {
  int cbBCWndIndex, cbBCWndExtra;

  if (nIndex < 0)
    return (nIndex);
  cbBCWndIndex = GetClassWord(hWnd, GCW_CBCLSEXTRA) - CBCLSEWNDINDEX;
  cbBCWndExtra = GetClassWord(hWnd, cbBCWndIndex);
  return (cbBCWndExtra + nIndex);
 }

WORD FAR PASCAL SetSCClassWord(HWND hWnd, int nIndex, WORD wValue)
 {
  nIndex = CalcClassIndex(hWnd, nIndex);
  return (SetClassWord(hWnd, nIndex, wValue));
 }

WORD FAR PASCAL GetSCClassWord(HWND hWnd, int nIndex)
 {
  nIndex = CalcClassIndex(hWnd, nIndex);
  return (GetClassWord(hWnd, nIndex));
 }

DWORD FAR PASCAL SetSCClassLong(HWND hWnd, int nIndex, DWORD dwValue)
 {
  nIndex = CalcClassIndex(hWnd, nIndex);
  return (SetClassLong(hWnd, nIndex, dwValue));
 }

DWORD FAR PASCAL GetSCClassLong(HWND hWnd, int nIndex)
```

```
 {
  nIndex = CalcClassIndex(hWnd, nIndex);
  return (GetClassLong(hWnd, nIndex));
 }

WORD FAR PASCAL SetSCWindowWord(HWND hWnd, int nIndex, WORD wValue)
 {
  nIndex = CalcWindowIndex(hWnd, nIndex);
  return (SetWindowWord(hWnd, nIndex, wValue));
 }

WORD FAR PASCAL GetSCWindowWord(HWND hWnd, int nIndex)
 {
  nIndex = CalcWindowIndex(hWnd, nIndex);
  return (GetWindowWord(hWnd, nIndex));
 }

DWORD FAR PASCAL SetSCWindowLong(HWND hWnd, int nIndex, DWORD dwValue)
 {
  nIndex = CalcWindowIndex(hWnd, nIndex);
  return (SetWindowLong(hWnd, nIndex, dwValue));
}

DWORD FAR PASCAL GetSCWindowLong(HWND hWnd, int nIndex)
 {
  nIndex = CalcWindowIndex(hWnd, nIndex);
  return (GetWindowLong(hWnd, nIndex));
 }
```

```
/*****************************************************************************
  MODULE: LongDlg.C

  PURPOSE: This file contain the functions to integrate the Long
           object into dialog boxes tools generator.

  FUNCTIONS:

  LongInfo()  - Return to Windows the name of class,
                the version and the title of object
  LongStyle() - Is called by dialog box tools for used in this
                the dialog box of specialization
  LongDlgFn() - Function to process the dlg messages
  LongFlags() - Build string to integrate in resources

*****************************************************************************/

#include <windows.h>
#include <custcntl.h>

#include "control.h"

#include "long.h"
#include "dialog.h"


/********** Definitions of constants ***************************************/


extern HANDLE _hInstance;
extern char   _szControlName[];


/********** Prototyping ****************************************************/


BOOL FAR PASCAL LongDlgFn (HWND hDlg, WORD wMessage, WORD wParam, LONG lParam);



GLOBALHANDLE FAR PASCAL LongInfo (void)
{
   GLOBALHANDLE hCtlInfo = NULL;

   hCtlInfo = ControlInfo(0x0100, _szControlName, _szControlName);
   if (hCtlInfo == NULL) return(hCtlInfo);
   AddControlType(hCtlInfo, 0, 40, 12, WS_CHILD | WS_TABSTOP | WS_BORDER | LGS_BEEP | LGS_STRICT, _szControlName)
   return(hCtlInfo);

}

BOOL FAR PASCAL LongStyle (HWND hWnd, GLOBALHANDLE hCtlStyle,
                           LPFNSTRTOID lpfnStrToId, LPFNIDTOSTR lpfnIdToStr)
 {
  return(ShowStyleDlg(_hInstance, "StyleDlg", hWnd, (FARPROC)LongDlgFn,
                      0, hCtlStyle, lpfnStrToId, lpfnIdToStr));
 }

BOOL FAR PASCAL LongDlgFn (HWND hDlg, WORD wMsg, WORD wParam, LONG lParam)
 {
  BOOL fResult = TRUE;
  char szStr[21];
  DWORD dwResult;
  LPCTLSTYLE lpCtlStyle;
```

```
switch (wMsg)
   {
    case WM_INITDIALOG:
         SendDlgItemMessage(hDlg, IDD_VALUE, EM_LIMITTEXT, 20, 0L);
         SendDlgItemMessage(hDlg, IDD_TEXT, EM_LIMITTEXT, 20, 0L);
         GetIdString(hDlg, szStr, sizeof(szStr));
         SetDlgItemText(hDlg, IDD_VALUE, szStr);
         lpCtlStyle = CtlStyleLock(hDlg);
         SetDlgItemText(hDlg, IDD_TEXT, lpCtlStyle->szTitle);
         SendDlgItemMessage(hDlg, IDD_BEEP, BM_SETCHECK,
                            (BOOL)(lpCtlStyle->dwStyle & LGS_BEEP), 0L);
         SendDlgItemMessage(hDlg, IDD_STRICT, BM_SETCHECK,
                            (BOOL)(lpCtlStyle->dwStyle & LGS_STRICT), 0L);
         SendDlgItemMessage(hDlg, IDD_PASSWORD, BM_SETCHECK,
                            (BOOL)(lpCtlStyle->dwStyle & ES_PASSWORD), 0L);
         SendDlgItemMessage(hDlg, IDD_NOHIDESEL, BM_SETCHECK,
                            (BOOL)(lpCtlStyle->dwStyle & ES_NOHIDESEL), 0L);
         SendDlgItemMessage(hDlg, IDD_OEM, BM_SETCHECK,
                            (BOOL)(lpCtlStyle->dwStyle & ES_OEMCONVERT), 0L);
         SendDlgItemMessage(hDlg, IDD_AUTOHSCROLL, BM_SETCHECK,
                            (BOOL)(lpCtlStyle->dwStyle & ES_AUTOHSCROLL), 0L);
         SendDlgItemMessage(hDlg, IDD_BORDER, BM_SETCHECK,
                            ((lpCtlStyle->dwStyle & WS_BORDER)?1:0), 0L);
         SendDlgItemMessage(hDlg, IDD_TABSTOP, BM_SETCHECK,
                            ((lpCtlStyle->dwStyle & WS_TABSTOP)?1:0), 0L);
         SendDlgItemMessage(hDlg, IDD_GROUP, BM_SETCHECK,
                            ((lpCtlStyle->dwStyle & WS_GROUP)?1:0), 0L);
         CtlStyleUnlock(hDlg);
         break;

    case WM_COMMAND:
       switch (wParam) {
          case IDOK:
               GetDlgItemText(hDlg, IDD_VALUE, szStr, sizeof(szStr));
               dwResult = SetIdValue(hDlg, szStr);
               if (LOWORD(dwResult) == 0) break;
               lpCtlStyle = CtlStyleLock(hDlg);
               lpCtlStyle->dwStyle &= 0xffff0000L;
               lpCtlStyle->dwStyle &= ~WS_BORDER;
               lpCtlStyle->dwStyle &= ~WS_TABSTOP;
               lpCtlStyle->dwStyle &= ~WS_GROUP;
               GetDlgItemText(hDlg, IDD_TEXT, szStr, sizeof(szStr));
               lstrcpy(lpCtlStyle->szTitle, (LPSTR)szStr);
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_BEEP, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= LGS_BEEP;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_STRICT, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= LGS_STRICT;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_PASSWORD, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= ES_PASSWORD;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_NOHIDESEL, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= ES_NOHIDESEL;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_AUTOHSCROLL, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= ES_AUTOHSCROLL;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_OEM, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= ES_OEMCONVERT;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_BORDER, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= WS_BORDER;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_TABSTOP, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= WS_TABSTOP;
               if ((BOOL)SendDlgItemMessage(hDlg, IDD_GROUP, BM_GETCHECK, 0, 0L))
                 lpCtlStyle->dwStyle |= WS_GROUP;
               CtlStyleUnlock(hDlg);
```

```
            case IDCANCEL:
                 EndDialog(hDlg, wParam);
                 break;

            case IDD_VALUE:
                 if (HIWORD(lParam) == EN_CHANGE)
                   EnableWindow(GetDlgItem(hDlg, IDOK),
                         SendMessage(LOWORD(lParam), WM_GETTEXTLENGTH, 0, 0L)
                         ? TRUE : FALSE);
                 break;

            default: fResult = FALSE;
                     break;
          }
          break;

       default: fResult = FALSE;
                break;
    }
    return(fResult);
}

WORD FAR PASCAL LongFlags (DWORD dwFlags, LPSTR szString, WORD wMaxString)
  {
   WORD x;
   *szString = 0;
   if (dwFlags & LGS_BEEP)
     lstrcat(szString, (LPSTR)"LGS_BEEP | ");
   if (dwFlags & LGS_STRICT)
     lstrcat(szString, (LPSTR)"LGS_STRICT | ");
   if (dwFlags & ES_PASSWORD)
     lstrcat(szString, (LPSTR)"ES_PASSWORD | ");
   if (dwFlags & ES_AUTOHSCROLL)
     lstrcat(szString, (LPSTR)"ES_AUTOHSCROLL | ");
   if (dwFlags & ES_NOHIDESEL)
     lstrcat(szString, (LPSTR)"ES_NOHIDESEL | ");
   if (dwFlags & ES_OEMCONVERT)
     lstrcat(szString, (LPSTR)"ES_OEMCONVERT | ");
   x = lstrlen(szString);
   if (x > 0) { x -= sizeof(" | ") - 1; *(szString + x) = 0; }
   return(x);
  }
```

```
/*****************************************************************************
  MODULE: Control.C

  PURPOSE: Contain the functions (generic) to interface object with dlg box
           editor.

  FUNCTIONS:


  ControlInfo()    - Allocate memory and stock info into structure
  AddControlType() - Add into dlg editor the control type of object
  ShowStyleDlg()   - Display in dlg editor the dialog box to select
                     style of object
  CtlStyleLock()   - Lock memory block which contain styles
  CtlStyleUnlock() - Unlock memory block which contain styles
  GetIdString()    - Call spcifique functions to convert string to an ID
  SetIdValue()     -  "      "          "

*****************************************************************************/

#include <windows.h>
#include <custcntl.h>

#include "control.h"


/********* Definition of structures and constants ***************************/


static char _szCtlProp[] = "CtlDlgStyleData";

typedef struct {
                GLOBALHANDLE hCtlStyle;
                LPFNSTRTOID  lpfnStrToId;
                LPFNIDTOSTR  lpfnIdToStr;
               } CTLSTYLEDLG, FAR *LPCTLSTYLEDLG, NEAR *NPCTLSTYLEDLG;




GLOBALHANDLE FAR PASCAL ControlInfo (WORD wVersion, LPSTR szClass, LPSTR szTitle)
 {
  GLOBALHANDLE hMem = NULL;
  LPCTLINFO    lpCtlInfo;

  hMem = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, (DWORD) sizeof(CTLINFO));
  if (hMem == NULL)
    return(hMem);
  lpCtlInfo = (LPCTLINFO) GlobalLock(hMem);
  lpCtlInfo->wVersion  = wVersion;

  lpCtlInfo->wCtlTypes = 0;
  lstrcpy(lpCtlInfo->szClass, szClass);
  lstrcpy(lpCtlInfo->szTitle, szTitle);
  GlobalUnlock(hMem);
  return(hMem);
 }

BOOL FAR PASCAL AddControlType (GLOBALHANDLE hMem, WORD wType, WORD wWidth,
                                WORD wHeight, DWORD dwStyle, LPSTR szDescr)
 {
  LPCTLINFO lpCtlInfo; WORD wNumTypes;
  lpCtlInfo = (LPCTLINFO) GlobalLock(hMem);
  wNumTypes = lpCtlInfo->wCtlTypes;
```

```
  if (wNumTypes == CTLTYPES)
   {
    GlobalUnlock(hMem);
    return(FALSE);
   }
  lpCtlInfo->Type[wNumTypes].wType   = wType;
  lpCtlInfo->Type[wNumTypes].wWidth  = wWidth;
  lpCtlInfo->Type[wNumTypes].wHeight = wHeight;
  lpCtlInfo->Type[wNumTypes].dwStyle = dwStyle;
  lstrcpy(lpCtlInfo->Type[wNumTypes].szDescr, szDescr);
  lpCtlInfo->wCtlTypes++;
  GlobalUnlock(hMem);
  return(TRUE);
 }

int FAR PASCAL ShowStyleDlg (HANDLE hInstance, LPSTR szTemplate,
                             HWND hWndParent, FARPROC fpDlgProc, LONG lParam,
                             GLOBALHANDLE hCtlStyle, LPFNSTRTOID lpfnStrToId,
                             LPFNIDTOSTR lpfnIdToStr)
 {
  LOCALHANDLE hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  int x;

  hCtlStyleDlg = LocalAlloc(LMEM_MOVEABLE | LMEM_ZEROINIT, sizeof(CTLSTYLEDLG));
  if (hCtlStyleDlg == NULL)
    return(FALSE);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  npCtlStyleDlg->hCtlStyle = hCtlStyle;
  npCtlStyleDlg->lpfnStrToId = lpfnStrToId;
  npCtlStyleDlg->lpfnIdToStr = lpfnIdToStr;
  LocalUnlock(hCtlStyleDlg);
  SetProp(hWndParent, _szCtlProp, hCtlStyleDlg);
  x = DialogBoxParam(hInstance, szTemplate, hWndParent, fpDlgProc, lParam);
  RemoveProp(hWndParent, _szCtlProp);
  LocalFree(hCtlStyleDlg);
  return(x == IDOK);
 }

LPCTLSTYLE FAR PASCAL CtlStyleLock (HWND hDlg)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle = NULL;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(lpCtlStyle);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  lpCtlStyle = (LPCTLSTYLE) GlobalLock(npCtlStyleDlg->hCtlStyle);
  LocalUnlock(hCtlStyleDlg);
  return(lpCtlStyle);
 }

BOOL FAR PASCAL CtlStyleUnlock (HWND hDlg)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  BOOL          fOk = FALSE;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(fOk);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
```

```
  fOk = GlobalUnlock(npCtlStyleDlg->hCtlStyle);
  LocalUnlock(hCtlStyleDlg);
  return(fOk);
 }

WORD FAR PASCAL GetIdString (HWND hDlg, LPSTR szId, WORD wIdMaxLen)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle;
  WORD          wIdLen;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(0);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  lpCtlStyle = (LPCTLSTYLE) GlobalLock(npCtlStyleDlg->hCtlStyle);
  wIdLen = (*npCtlStyleDlg->lpfnIdToStr)(lpCtlStyle->wId, szId, wIdMaxLen);
  GlobalUnlock(npCtlStyleDlg->hCtlStyle);
  LocalUnlock(hCtlStyleDlg);
  return(wIdLen);
 }

DWORD FAR PASCAL SetIdValue (HWND hDlg, LPSTR szId)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle;
  DWORD         dwResult = 0;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(dwResult);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  dwResult = (*npCtlStyleDlg->lpfnStrToId)(szId);
  LocalUnlock(hCtlStyleDlg);
  if (LOWORD(dwResult) == 0)
    return(dwResult);
  lpCtlStyle = CtlStyleLock(hDlg);
  lpCtlStyle->wId = HIWORD(dwResult);
  CtlStyleUnlock(hDlg);
  return(dwResult);
 }
```

```
/*****************************************************************************
  MODULE: Util.c

  PURPOSE: Utilities for  manipulation of strings

  FUNCTIONS:

  Notification()        - Notification of parent
  lCharInString()       - Check if one character is in string
  lInsertCharInString() - Insert a positionned char in string
  lDeleteCharInString() - Delete a positionned char in string
  lAllCharInString()    - Check in all char are in string

*****************************************************************************/

#include <windows.h>

#include "util.h"


void FAR PASCAL Notification(HWND hWnd, WORD wCode)
 {
  SendMessage(GetParent(hWnd), WM_COMMAND,
              GetWindowWord(hWnd, GWW_ID),
              MAKELONG(hWnd, wCode));
 }

int FAR PASCAL lCharInString(char car, LPSTR lpString)
 {
  WORD wCurrent, wSize;

  wSize = lstrlen(lpString);
  for (wCurrent = 0; wCurrent < wSize; wCurrent++)
   {
    if (lpString[wCurrent] == car)
     return (++wCurrent);
   }
  return (0);
 }

void FAR PASCAL lInsertCharInString(LPSTR lpString, WORD wPos, char car)
 {
  WORD wCurrent, wSize;

  wSize = lstrlen(lpString);
  lpString[wSize+1] = '\0';
  for (wCurrent = wSize; wCurrent > wPos; wCurrent--)
   lpString[wCurrent] = lpString[wCurrent-1];
  lpString[wPos] = car;
 }

void FAR PASCAL lDeleteCharInString(LPSTR lpString, WORD wPos, WORD wNb)
 {
  WORD wCurrent, wSize;

  wSize = lstrlen(lpString) - wPos - wNb;
  for (wCurrent = wPos; wCurrent < wSize+wPos; wCurrent++)
    lpString[wCurrent] = lpString[wCurrent + wNb];
  lpString[wSize + wPos] = '\0';
 }

BOOL FAR PASCAL lAllCharInString(char car, LPSTR lpString)
 {
  WORD wSize, wCurrent;
```

```
 for (wCurrent = 0; wCurrent < wSize; wCurrent++)
  if (lpString[wCurrent] != car)
    return (FALSE);
 return (TRUE);
}
```

```
/*****************************************************************************
File name: Long.H
*****************************************************************************/

// Long control's class-specific window styles.
#define LGS_BEEP              0x2000L
#define LGS_STRICT            0x4000L

// Long control's class-specific window messages.
#define LGM_SETMINVALUE       (WM_USER + 50)
#define LGM_GETMINVALUE       (WM_USER + 51)
#define LGM_SETMAXVALUE       (WM_USER + 52)
#define LGM_GETMAXVALUE       (WM_USER + 53)
#define LGM_SETCURRENTVALUE   (WM_USER + 54)
#define LGM_GETCURRENTVALUE   (WM_USER + 55)
#define LGM_CHECKCURRENTVALUE (WM_USER + 56)
#define LGM_SETALLVALUES      (WM_USER + 57)

// Long control's notification codes to send to parent.
#define LGN_OUTOFRANGE 0x1000
#define LGN_BADCHAR    0x2000
#define LGN_BADPOSCHAR 0x3000
#define LGN_BADFORMAT  0x4000
#define LGN_ERROR      0x5000

// Long control's error codes
#define LG_OUTOFRANGE  (-2)
#define LG_BADFORMAT   (-1)
#define LG_ERR          (0)
#define LG_NULL         (1)
#define LG_OK           (2)

// Long control's Minimum and maximum values
#define MINLONG        0x80000000L
#define MAXLONG        0x7FFFFFFFL

// Long control's prototypes

LONG FAR PASCAL LongAddress(LONG);

typedef LONG (FAR PASCAL *LONGADDRESS)(LONG);
```

```
#define IDD_VALUE         100
#define IDD_TEXT          101
#define IDD_BEEP          102
#define IDD_STRICT        103
#define IDD_OEM           104
#define IDD_PASSWORD      105
#define IDD_NOHIDESEL     106
#define IDD_AUTOHSCROLL   107
#define IDD_TABSTOP       108
#define IDD_GROUP         109
#define IDD_BORDER        110
```

```
/*****************************************************************************
File name: Control.H
*****************************************************************************/

GLOBALHANDLE FAR PASCAL ControlInfo (WORD wVersion, LPSTR szClass, LPSTR szTitle);
BOOL FAR PASCAL AddControlType (GLOBALHANDLE hMem, WORD wType, WORD wWidth,
                                WORD wHeight, DWORD dwStyle, LPSTR szDescr);
int FAR PASCAL ShowStyleDlg (HANDLE hInstance, LPSTR szTemplate,
                             HWND hWndParent, FARPROC fpDlgProc, LONG lParam,
                             GLOBALHANDLE hCtlStyle, LPFNSTRTOID lpfnStrToId,
                             LPFNIDTOSTR lpfnIdToStr);
LPCTLSTYLE FAR PASCAL CtlStyleLock (HWND hDlg);
BOOL FAR PASCAL CtlStyleUnlock (HWND hDlg);
WORD FAR PASCAL GetIdString (HWND hDlg, LPSTR szId, WORD wIdMaxLen);
DWORD FAR PASCAL SetIdValue (HWND hDlg, LPSTR szId);
```

```
/*****************************************************************************
File name: SuperCls.H
*****************************************************************************/

BOOL FAR PASCAL RegisterSuperClass(LPWNDCLASS lpWC, LONG (FAR PASCAL *lpfnSCWndProc)(), int cbClsAdditional, int
void FAR PASCAL SetSuperClassInfo(HWND);
DWORD FAR PASCAL GetBCWndProc(HWND);
WORD FAR PASCAL SetSCClassWord(HWND, int, WORD);
WORD FAR PASCAL GetSCClassWord(HWND, int);
DWORD FAR PASCAL SetSCClassLong(HWND, int, DWORD);
DWORD FAR PASCAL GetSCClassLong(HWND, int);
WORD FAR PASCAL SetSCWindowWord(HWND, int, WORD);
WORD FAR PASCAL GetSCWindowWord(HWND, int);
DWORD FAR PASCAL SetSCWindowLong(HWND, int, DWORD);
DWORD FAR PASCAL GetSCWindowLong(HWND, int);
```

```
/*****************************************************************************
File name: Util.h
*****************************************************************************/

void FAR PASCAL Notification(HWND, WORD);
int FAR PASCAL lCharInString(char, LPSTR);
void FAR PASCAL lInsertCharInString(LPSTR, WORD, char);
void FAR PASCAL lDeleteCharInString(LPSTR, WORD, WORD);
BOOL FAR PASCAL lAllCharInString(char, LPSTR);
```

```
STYLEDLG DIALOG  50,0,140,133
STYLE WS_POPUP | WS_CAPTION
CAPTION "Long Style..."
BEGIN
    CONTROL "&ID Value:",-1,"Static",WS_CHILD | WS_VISIBLE | SS_LEFT,10,10,36,8
    CONTROL "",IDD_VALUE,"Edit",ES_AUTOHSCROLL | WS_GROUP | WS_TABSTOP | WS_CHILD | WS_VISIBLE | WS_BORDER | ES_L
    CONTROL "&Value:",100,"STATIC",WS_CHILD | SS_LEFT,10,25,36,8
    CONTROL "",IDD_TEXT,"Edit",ES_AUTOHSCROLL | WS_GROUP | WS_TABSTOP | WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LE
    CONTROL "Style",-1,"button",BS_GROUPBOX | WS_GROUP | WS_CHILD,5,35,130,49
    CONTROL "&Auto Horz. Scroll",IDD_AUTOHSCROLL,"Button",BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_CHILD | WS
    CONTROL "No &Hide Selection",IDD_NOHIDESEL,"Button",BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE,9,59,70,10
    CONTROL "&OEM_Convert",IDD_OEM,"Button",BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE,9,71,70,10
    CONTROL "&Password",IDD_PASSWORD,"Button",BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE,84,47,45,10
    CONTROL "B&eep",IDD_BEEP,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD,84,59,45,10
    CONTROL "&Strict entry",IDD_STRICT,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD,84,71,47,10
    CONTROL "&Tabstop",IDD_TABSTOP,"button",BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_CHILD,10,90,38,10
    CONTROL "&Border",IDD_BORDER,"Button",BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_CHILD | WS_VISIBLE,53,90,3
    CONTROL "&Group",IDD_GROUP,"button",BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_CHILD,96,90,38,10
    CONTROL "&OK",IDOK,"Button",BS_DEFPUSHBUTTON | WS_GROUP | WS_TABSTOP | WS_CHILD | WS_VISIBLE,26,108,32,14
    CONTROL "&Cancel",IDCANCEL,"Button",WS_GROUP | WS_TABSTOP | WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,83,108,32,1
END
```

```
; File name: Long.def

LIBRARY     LONG
DESCRIPTION 'Long Custom Control Library'
EXETYPE     WINDOWS
STUB        'WinStub.Exe'

CODE        MOVEABLE DISCARDABLE SHARED PRELOAD
DATA        MOVEABLE SINGLE PRELOAD

HEAPSIZE    1024

EXPORTS
   WEP            @1    RESIDENTNAME
   LongInfo       @2
   LongStyle      @3
   LongFlags      @4
   LongWndFn      @5
   LongDlgFn      @6
   LongAddress    @7
```

```
/*****************************************************************************
File name: Long.RC
*****************************************************************************/

#include <windows.h>

#include "long.h"
#include "dialog.h"
#include "long.dlg"
```

```
#*****************************************************************************
# File name: MAKEFILE
#*****************************************************************************

PROG  = Long
MODEL = S
CFLAGS = -A$(MODEL)w -D_WINDOWS -D_WINDLL -Gcsw2 -W4 -Zlepid -Od
LFLAGS = /NOE/BA/A:16/M/CO/LI/F
LIBS = $(MODEL)nocrtd + $(MODEL)dllcew + libw

M1 = $(PROG).obj $(PROG)Dlg.obj util.obj Control.obj SuperCls.obj

ICONS   =
BITMAPS =
CURSORS =
RESOURCES = $(ICONS) $(BITMAPS) $(CURSORS)

.SUFFIXES: .rc

.rc.res:
    rc -r $*.rc

$(PROG).LIB: $(PROG).DLL $(PROG).DEF
      implib $(PROG).lib $(PROG).def
      copy $(PROG).lib c:\windev\lib

$(PROG).DLL: $(M1) $(PROG).Def $(PROG).Res
     link $(LFLAGS) @<<$(PROG).Lnk
C:\WINDEV\LIB\LibEntry.obj $(M1)
$(PROG).DLL, $(PROG), $(LIBS), $(PROG)
<<
     rc -Fe$(PROG).DLL $(PROG).Res
     copy $(PROG).dll c:\win3

$(PROG).obj:    $*.c $*.h
$(PROG)Dlg.obj: $*.c $(PROG).h  dialog.h
Control.obj:    $*.c $*.h
Util.obj:       $*.c $*.h
SuperCls.obj:   $*.c $*.h
$(PROG).res:    $*.rc $*.h $*.dlg dialog.h $(RESOURCES)
```

3. Objet interactif de type "GraphicButton".

```
/*****************************************************************************
  MODULE : GButton.C

  PURPOSE : Make and manage all model of buttons with graphics images
            in remplacement of classical text.

  FUNCTIONS :

  RegisterControlClass() - Register Graphics button control class
  lread()                - Read Bitmap that size can be upper 64 K
  NumColors()            - Return Colors type used in a bitmap
  IsTypeButton()         - Return true if object is a button
  PrepareBitmap()        - Retreive and file Bitmap into handle
  GetValidBrush()        - Ask to parent to change apparence
  PaintFrameGButton()    - Paint frame of graphic button
  GetBitmapPos()         - Retreive the position of bitmap
  BorderGButton()        - Paint the border of button
  FocusGButton()         - Draw focus frame around the button
  DisabledGButton()      - Draw the button in inactive mode
  PaintGButton()         - Paint the bitmap in button
  UpdateCheckInGroup()   - Update autoradio Buttons
  SizeGButton()          - Set size of button in function of bitmap and mode used
  MoveGButton()          - Move bitmap in button when it is activate
  GraphicButtonWndFn()   - Process window message (wnd function for graphic button)


*****************************************************************************/

#include <windows.h>
#include <math.h>

#include "gbutton.h"
#include "private.h"

/********** Definition of structures & constants ***************************/

HANDLE _hInstance       = NULL;
char   _szControlName[] = "GraphicButton";

#ifdef DIALOG_EDITOR_30
DWORD  _dwStyle         = 0;
#endif

#define POSBORDER   (1)
#define POSFOCUS    (4)

#define SIZEOFFILE 13    // sizeof filename with extension
#define MAXREAD    32768

#define SELECTED      0x0001
#define HASFOCUS      0x0002
#define KEYDOWN       0x0004
#define MOUSEDOWN     0x0008
#define MOUSEACTIVATE 0x0010

/********** Definition of control extra bytes ******************************/

#define CBWNDEXTRA         (12)

#define GetHBitmap        GetWindowWord(hWnd, 0)
#define GetPrivate        GetWindowWord(hWnd, 2)
#define GetLeftTop        GetWindowLong(hWnd, 4)
#define GetRightBottom    GetWindowLong(hWnd, 8)
```

```
#define SetHBitmap(x)     SetWindowWord(hWnd, 0, x)
#define SetPrivate(x)     SetWindowWord(hWnd, 2, x)
#define SetLeftTop(x)     SetWindowLong(hWnd, 4, x)
#define SetRightBottom(x) SetWindowLong(hWnd, 8, x)

#define GetStyle          GetWindowLong(hWnd, GWL_STYLE)
#define SetStyle(x)       SetWindowLong(hWnd, GWL_STYLE, x)


/********** Prototyping ************************************************************/

static short NEAR PASCAL RegisterControlClass (HANDLE hInstance);
LONG FAR PASCAL GraphicButtonWndFn (HWND hWnd, WORD wMsg, WORD wParam, LONG lParam);

/********** Window's Dynamic-Link Library Initialization Routines ************/

BOOL FAR PASCAL LibMain (HANDLE hModule, WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdLine)
 {
  if (_hInstance == NULL)
   {
    if (wHeapSize != 0)
      UnlockData(0);
    _hInstance = (RegisterControlClass(hModule) ? hModule : NULL);
   }
  return(_hInstance ? TRUE : FALSE);
 }

VOID FAR PASCAL WEP (int nSystemExit)
 {
  return;
 }

static short NEAR PASCAL RegisterControlClass (HANDLE hInstance)
 {
  WNDCLASS wc;
  wc.style         = CS_GLOBALCLASS | CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
  wc.lpfnWndProc   = GraphicButtonWndFn;
  wc.cbClsExtra    = 0;
  wc.cbWndExtra    = CBWNDEXTRA;
  wc.hInstance     = hInstance;
  wc.hIcon         = NULL;
  wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(LTGRAY_BRUSH);
  wc.lpszMenuName  = NULL;
  wc.lpszClassName = _szControlName;
  return(RegisterClass(&wc));
 }

static DWORD NEAR PASCAL lread (int fh, VOID FAR *pv, DWORD ul)
 {
  DWORD     ulT = ul;
  BYTE huge *hp = pv;

  while (ul > MAXREAD)
   {
    if (_lread(fh, (LPSTR)hp, (WORD)MAXREAD) != MAXREAD)
      return 0;
    ul -= MAXREAD;
    hp += MAXREAD;
   }
  if (_lread(fh, (LPSTR)hp, (WORD)ul) != (WORD)ul)
    return 0;
  return ulT;
 }
```

```
static WORD NEAR PASCAL NumColors(LPBITMAPINFOHEADER lpBiInfoHeader)
 {
  if (lpBiInfoHeader->biClrUsed != 0)
    return (WORD)lpBiInfoHeader->biClrUsed;

  switch (lpBiInfoHeader->biBitCount)
       {
        case 1:
                return 2;
        case 4:
                return 16;
        case 8:
                return 256;
        default:
                return 0;
       }
 }

static BOOL NEAR PASCAL IsTypeButton(HWND hWnd, WORD wType)
 {
  if (wType & SendMessage(hWnd, WM_GETDLGCODE, 0, 0L))
    return TRUE;
  else
    return FALSE;
 }

static VOID NEAR PASCAL PrepareBitmap(HWND hWnd, LPSTR szFilename, short FAR *shWidth, short FAR *shHeight)
 {
  HBITMAP          hBitmap;
  HDC              hDC;
  BITMAP           bitmap;
  OFSTRUCT         ofStruct;
  BITMAPFILEHEADER bitmapFileHeader;
  BITMAPINFOHEADER bitmapInfoHeader;
  GLOBALHANDLE     hBitmapInfo;
  LPBITMAPINFO     lpBitmapInfo;
  HANDLE           hBit;
  LPSTR            lpBits;
  HANDLE           hFile;
  WORD             wClrs;
  DWORD            dwSize;

  hDC = GetDC(hWnd);
  if ((GetDeviceCaps(hDC,RASTERCAPS) & RC_BITBLT) != RC_BITBLT)
   {
    ReleaseDC(hWnd, hDC);
    return ;
   }
  ReleaseDC(hWnd, hDC);
  if (!(hBitmap = LoadBitmap(_hInstance, szFilename)))
   {
    lstrcat((LPSTR)szFilename, (LPSTR)".BMP");
    if ((hFile = OpenFile( szFilename, &ofStruct, OF_READ)) == -1)
      return ;
    if (lread(hFile, (LPSTR)&bitmapFileHeader, sizeof(BITMAPFILEHEADER))
        != sizeof(bitmapFileHeader))
     {
      _lclose(hFile);
      return ;
     }
    if (bitmapFileHeader.bfType != 0x4d42)
     {
      _lclose(hFile);
```

```
      return ;
     }
    if (lread(hFile, (LPSTR)&bitmapInfoHeader, sizeof(BITMAPINFOHEADER))
        != sizeof(bitmapInfoHeader))
     {
      _lclose(hFile);
      return ;
     }
    wClrs = NumColors(&bitmapInfoHeader);
    hBitmapInfo = GlobalAlloc(GHND, sizeof(BITMAPINFOHEADER) + (sizeof(RGBQUAD) * wClrs));
    lpBitmapInfo = (LPBITMAPINFO)GlobalLock(hBitmapInfo);
    lpBitmapInfo->bmiHeader = bitmapInfoHeader;
    if (lread(hFile, (LPSTR)&lpBitmapInfo->bmiColors[0], sizeof(RGBQUAD) * wClrs)
        != sizeof(RGBQUAD) * wClrs)
     {
      GlobalUnlock(hBitmapInfo);
      GlobalFree(hBitmapInfo);
      _lclose(hFile);
      return ;
     }
    dwSize = bitmapFileHeader.bfSize - sizeof(BITMAPFILEHEADER) -
             sizeof(BITMAPINFOHEADER) - (sizeof(RGBQUAD) * wClrs);
    hBit = GlobalAlloc(GHND, dwSize);
    lpBits = (LPSTR)GlobalLock(hBit);
    if (lread(hFile, lpBits, dwSize) != dwSize)
     {
      GlobalUnlock(hBitmapInfo);
      GlobalFree(hBitmapInfo);
      GlobalUnlock(hBit);
      GlobalFree(hBit);
      _lclose(hFile);
      return ;
     }
    _lclose(hFile);
    hBitmap = CreateDIBitmap(hDC, &bitmapInfoHeader, CBM_INIT, lpBits, lpBitmapInfo, DIB_RGB_COLORS);
    GlobalUnlock(hBit);
    GlobalFree(hBit);
    GlobalUnlock(hBitmapInfo);
    GlobalFree(hBitmapInfo);
   }

  if (!GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&bitmap))
   {
    DeleteObject(hBitmap);
    return ;
   }
  *shWidth = bitmap.bmWidth;
  *shHeight = bitmap.bmHeight;
  SetHBitmap(hBitmap);
 }


static VOID NEAR PASCAL GetValidBrush(HWND hWnd, HDC hDC, HBRUSH FAR *hBrush, DWORD FAR *dwColor)
 {
  HBRUSH   hlBrush = 0;
  LOGBRUSH logbrush;

  hlBrush = (HBRUSH) SendMessage(GetParent(hWnd), WM_CTLCOLOR, hDC,
                                 MAKELONG(hWnd, CTLCOLOR_GBUTTON));
  if (!GetObject(hlBrush, sizeof(LOGBRUSH), (LPSTR)&logbrush) ||
      logbrush.lbStyle != BS_SOLID || logbrush.lbColor == RGB(255,255,255))
  {
   hlBrush = GetClassWord(hWnd, GCW_HBRBACKGROUND);
   GetObject(hlBrush, sizeof(LOGBRUSH), (LPSTR)&logbrush);
```

```
  }
 if (hBrush)
   *hBrush = hlBrush;
 if (dwColor)
  *dwColor = logbrush.lbColor;
 }

static VOID NEAR PASCAL PaintFrameGButton(HWND hWnd, HDC hDC)
 {
  HBRUSH  hBrush;
  RECT    rect;

  GetValidBrush(hWnd, hDC, &hBrush, NULL);
  GetClientRect(hWnd, &rect);
  FillRect(hDC, &rect, hBrush);
 }

static VOID NEAR PASCAL GetBitmapPos(HWND hWnd, LPRECT lpRect)
 {
  lpRect->left = (int)LOWORD(GetLeftTop);
  lpRect->top = (int)HIWORD(GetLeftTop);
  lpRect->right = (int)LOWORD(GetRightBottom);
  lpRect->bottom = (int)HIWORD(GetRightBottom);
 }

static VOID NEAR PASCAL BorderGButton(HWND hWnd, HDC hDC)
 {
  RECT    rect;
  HPEN    hPen;
  HBRUSH  hBrush;
  POINT   pTab[6];
  DWORD   dwColor;

  GetClientRect(hWnd, &rect);
  if (IsTypeButton(hWnd, DLGC_DEFPUSHBUTTON))
   {
    hPen = SelectObject(hDC, GetStockObject(BLACK_PEN));
    hBrush = SelectObject(hDC, GetStockObject(NULL_BRUSH));
    Rectangle(hDC, rect.left, rect.top, rect.right, rect.bottom);
    SelectObject(hDC, hPen);
    SelectObject(hDC, hBrush);
    SetRect(&rect, rect.left+1, rect.top+1, rect.right-1, rect.bottom-1);
   }
  GetValidBrush(hWnd, hDC, &hBrush, &dwColor);
  if (GetPrivate & SELECTED)
   {
    hPen = SelectObject(hDC, CreatePen(PS_SOLID, 0, RGB(128, 128, 128)));
    SelectObject(hDC, GetStockObject(GRAY_BRUSH));
    pTab[0].x = rect.left;
    pTab[0].y = rect.top;
    pTab[1].x = rect.right-rect.left;
    pTab[1].y = rect.top;
    pTab[2].x = rect.right-rect.left;
    pTab[2].y = rect.top+POSBORDER;
    pTab[3].x = rect.left+POSBORDER;
    pTab[3].y = rect.top+POSBORDER;
    pTab[4].x = rect.left+POSBORDER;
    pTab[4].y = rect.bottom-rect.top;
    pTab[5].x = rect.left;
    pTab[5].y = rect.bottom-rect.top;
    Polygon(hDC, pTab, 6);

    DeleteObject(SelectObject(hDC, CreatePen(PS_SOLID, 0, dwColor)));
    SelectObject(hDC, hBrush);
```

```
    pTab[0].x = rect.right-rect.left;
    pTab[0].y = POSBORDER+1+rect.top;
    pTab[1].x = rect.right-rect.left;
    pTab[1].y = rect.bottom-rect.top;
    pTab[2].x = POSBORDER+1+rect.left;
    pTab[2].y = rect.bottom-rect.top;
    pTab[3].x = POSBORDER+1+rect.left;
    pTab[3].y = rect.bottom-POSBORDER-1;
    pTab[4].x = rect.right-POSBORDER-1;
    pTab[4].y = rect.bottom-POSBORDER-1;
    pTab[5].x = rect.right-POSBORDER-1;
    pTab[5].y = rect.top+2*POSBORDER;
    Polygon(hDC, pTab, 6);
    DeleteObject(SelectObject(hDC, hPen));
   }
  else
   {
    SelectObject(hDC, GetStockObject(WHITE_PEN));
    SelectObject(hDC, GetStockObject(WHITE_BRUSH));
    SetPolyFillMode(hDC, WINDING);
    pTab[0].x = rect.left;
    pTab[0].y = rect.top;
    pTab[1].x = rect.right-rect.left;
    pTab[1].y = rect.top;
    pTab[2].x = rect.right-POSBORDER;
    pTab[2].y = POSBORDER+rect.top;
    pTab[3].x = POSBORDER+rect.left;
    pTab[3].y = POSBORDER+rect.top;
    pTab[4].x = POSBORDER+rect.left;
    pTab[4].y = rect.bottom-POSBORDER;
    pTab[5].x = rect.left;
    pTab[5].y = rect.bottom-rect.top;
    Polygon(hDC, pTab, 6);

    hPen = SelectObject(hDC, CreatePen(PS_SOLID, 0, RGB(128, 128, 128)));
    SelectObject(hDC, GetStockObject(GRAY_BRUSH));
    pTab[0].x = rect.right-rect.left;
    pTab[0].y = rect.top;
    pTab[1].x = rect.right-rect.left;
    pTab[1].y = rect.bottom-rect.top;
    pTab[2].x = rect.left;
    pTab[2].y = rect.bottom-rect.top;
    pTab[3].x = POSBORDER+1+rect.left;
    pTab[3].y = rect.bottom-POSBORDER-1;
    pTab[4].x = rect.right-POSBORDER-1;
    pTab[4].y = rect.bottom-POSBORDER-1;
    pTab[5].x = rect.right-POSBORDER-1;
    pTab[5].y = POSBORDER+1+rect.top;
    Polygon(hDC, pTab, 6);
    DeleteObject(SelectObject(hDC, hPen));
   }
 }

static VOID NEAR PASCAL FocusGButton(HWND hWnd, HDC hDC)
 {
  RECT rect;

  if (GetPrivate & HASFOCUS)
   {
    GetBitmapPos(hWnd, &rect);
    if (GetPrivate & SELECTED)
     {
      SetRect(&rect, rect.left-POSFOCUS+POSBORDER, rect.top-POSFOCUS+POSBORDER,
                     rect.left+rect.right+POSFOCUS+POSBORDER,
```

```
                    rect.top+rect.bottom+POSFOCUS+POSBORDER);
      DrawFocusRect(hDC, &rect);
     }
    else
     {
      SetRect(&rect, rect.left-POSFOCUS, rect.top-POSFOCUS,
                    rect.left+rect.right+POSFOCUS,
                    rect.top+rect.bottom+POSFOCUS);
      DrawFocusRect(hDC, &rect);
     }
   }
 }

static VOID NEAR PASCAL DisabledGButton(HWND hWnd, HDC hDC)
 {
  WORD   wMask[] = {0xAA,0x55,0xAA,0x55,0xAA,0x55,0xAA,0x55};
  RECT   rect;
  HBRUSH hBrush;
  HBITMAP hBitmap;

  GetBitmapPos(hWnd, (LPRECT)&rect);
  hBitmap = CreateBitmap(8,8,1,1,(LPSTR)wMask);
  hBrush = CreatePatternBrush(hBitmap);
  hBrush = SelectObject(hDC, hBrush);
  if (GetStyle & GBS_OR)
    PatBlt(hDC, rect.left, rect.top, rect.right, rect.bottom, 0xFA0089);
  else
    PatBlt(hDC, rect.left, rect.top, rect.right, rect.bottom, 0xF50225);
  DeleteObject(SelectObject(hDC, hBrush));
  DeleteObject(hBitmap);
 }

static VOID NEAR PASCAL PaintGButton(HWND hWnd, HDC hDC)
 {
  HDC     hDCMem;
  HBITMAP hBitmap;
  BITMAP  bitmap;
  POINT   pt;

  if (hBitmap = GetHBitmap)
   {
    hDCMem = CreateCompatibleDC(hDC);
    SelectObject(hDCMem, hBitmap);
    GetObject(hBitmap, sizeof(BITMAP), (LPSTR)&bitmap);
    SetMapMode(hDCMem, GetMapMode(hDC));
    pt.x = 0;
    pt.y = 0;
    DPtoLP(hDC, &pt, 1);
    if (GetStyle & GBS_CLIP || GetStyle & GBS_RESIZE)
      BitBlt(hDC, LOWORD(GetLeftTop) + (GetPrivate & SELECTED ? POSBORDER : 0),
                  HIWORD(GetLeftTop) + (GetPrivate & SELECTED ? POSBORDER : 0),
                  LOWORD(GetRightBottom), HIWORD(GetRightBottom),
                  hDCMem, pt.x, pt.y, SRCCOPY);
    else
     StretchBlt(hDC, LOWORD(GetLeftTop) + (GetPrivate & SELECTED ? POSBORDER : 0),
                     HIWORD(GetLeftTop) + (GetPrivate & SELECTED ? POSBORDER : 0),
                     LOWORD(GetRightBottom), HIWORD(GetRightBottom),
                     hDCMem, pt.x, pt.y,
                     bitmap.bmWidth, bitmap.bmHeight, SRCCOPY);
    DeleteDC(hDCMem);
   }
 }

static VOID NEAR PASCAL UpdateCheckInGroup(HWND hWnd)
```

```
 {
  HWND hCurrent;

  hCurrent = GetNextDlgGroupItem(GetParent(hWnd), hWnd, TRUE);
  while (hCurrent != NULL && hCurrent != hWnd)
   {
    if (IsTypeButton(hCurrent, DLGC_RADIOBUTTON))
      SendMessage(hCurrent, BM_SETCHECK, FALSE, 0L);
    hCurrent = GetNextDlgGroupItem(GetParent(hWnd), hCurrent, TRUE);
   }
 }

static VOID NEAR PASCAL SizeGButton(HWND hWnd, LONG lSize)
 {
  RECT    rect;
  HBITMAP hBitmap;
  BITMAP  bitmap;

  if (hBitmap = GetHBitmap)
   {
    GetObject(hBitmap, sizeof(BITMAP), (LPSTR)&bitmap);
    if (GetStyle & GBS_CLIP || GetStyle & GBS_RESIZE)
      SetRect(&rect, POSBITMAP+(bitmap.bmWidth < (int)LOWORD(lSize)-(2*POSBITMAP) ?
                     (LOWORD(lSize)-(2*POSBITMAP)-bitmap.bmWidth)/2: 0),
                     POSBITMAP+(bitmap.bmHeight < (int)HIWORD(lSize)-(2*POSBITMAP) ?
                     (HIWORD(lSize)-(2*POSBITMAP)-bitmap.bmHeight)/2: 0),
                     bitmap.bmWidth < (int)LOWORD(lSize) - (2*POSBITMAP) ?
                     bitmap.bmWidth : (int)LOWORD(lSize) - (2*POSBITMAP),
                     bitmap.bmHeight < (int)HIWORD(lSize) - (2*POSBITMAP) ?
                     bitmap.bmHeight : (int)HIWORD(lSize) - (2*POSBITMAP));
    else
      SetRect(&rect, POSBITMAP, POSBITMAP,
                     LOWORD(lSize)-(2*POSBITMAP), HIWORD(lSize)-(2*POSBITMAP));
   }
  else
    SetRect(&rect, POSBITMAP, POSBITMAP,
                   LOWORD(lSize)-(2*POSBITMAP), HIWORD(lSize)-(2*POSBITMAP));
  SetLeftTop(MAKELONG(rect.left, rect.top));
  SetRightBottom(MAKELONG(rect.right, rect.bottom));
 }

static VOID NEAR PASCAL MoveGButton(HWND hWnd)
 {
  RECT rect;

  GetBitmapPos(hWnd, &rect);
  if (GetPrivate & SELECTED)
   {
    SetRect(&rect, rect.left-POSFOCUS, rect.top-POSFOCUS,
                   rect.left+rect.right+POSFOCUS,
                   rect.top+rect.bottom+POSFOCUS);
    ScrollWindow(hWnd, POSBORDER, POSBORDER, &rect, &rect);
   }
  else
   {
    SetRect(&rect, rect.left-POSFOCUS+POSBORDER, rect.top-POSFOCUS+POSBORDER,
                   rect.left+rect.right+POSFOCUS+POSBORDER,
                   rect.top+rect.bottom+POSFOCUS+POSBORDER);
    ScrollWindow(hWnd, -POSBORDER, -POSBORDER, &rect, &rect);
   }
 }

LONG FAR PASCAL GraphicButtonWndFn (HWND hWnd, WORD wMsg, WORD wParam, LONG lParam)
 {
```

```
  LONG          lResult = 0;
  PAINTSTRUCT   ps;
  RECT          rect;
  char          szFilename[SIZEOFFILE];
  short         shWidth, shHeight;
  HDC           hDC;

  switch (wMsg)
    {
     case WM_GETDLGCODE:
          lResult = DLGC_BUTTON;
          if (!((DWORD)GetStyle & GBS_DEFPUSHBUTTON) &&
              !((DWORD)GetStyle & GBS_RADIOBUTTON) &&
              !((DWORD)GetStyle & GBS_AUTORADIOBUTTON) &&
              !((DWORD)GetStyle & GBS_CHECKBOX) &&
              !((DWORD)GetStyle & GBS_AUTOCHECKBOX))
           {
            lResult = lResult | DLGC_UNDEFPUSHBUTTON;
            break;
           }
          if ((DWORD)GetStyle & GBS_DEFPUSHBUTTON)
           {
            lResult = lResult | DLGC_DEFPUSHBUTTON;
            break;
           }
          if ((DWORD)GetStyle & GBS_RADIOBUTTON ||
              (DWORD)GetStyle & GBS_AUTORADIOBUTTON)
            lResult = lResult | DLGC_RADIOBUTTON;
          break;

     case WM_CREATE:
          SendMessage(hWnd, WM_GETTEXT, sizeof(szFilename), (LONG)(LPSTR)szFilename);
          PrepareBitmap(hWnd, (LPSTR)szFilename, &shWidth, &shHeight);
#ifdef DIALOG_EDITOR_30
          if (_dwStyle)   // this global variable is neccessary to update via Dialog box editor
           {
            SetStyle(_dwStyle);
            _dwStyle = 0;
           }
#endif
          if (GetHBitmap && GetStyle & GBS_RESIZE)
            SetWindowPos(hWnd, NULL, 0, 0, shWidth+(2*POSBITMAP), shHeight+(2*POSBITMAP), SWP_NOMOVE | SWP_NOZORD
          lResult = TRUE;
          break;

     case WM_SETTEXT:
          lResult = DefWindowProc(hWnd, wMsg, wParam, lParam);
          if (GetHBitmap)
            DeleteObject(GetHBitmap);
          PrepareBitmap(hWnd, (LPSTR)lParam, &shWidth, &shHeight);
          if (GetHBitmap && GetStyle & GBS_RESIZE)
            SetWindowPos(hWnd, NULL, 0, 0, shWidth+(2*POSBITMAP), shHeight+(2*POSBITMAP), SWP_NOMOVE | SWP_NOZORD
          else
           {
            GetClientRect(hWnd, &rect); // to update pos of bitmap
            SetWindowPos(hWnd, NULL, 0, 0, rect.right, rect.bottom, SWP_NOMOVE | SWP_NOZORDER);
           }
          InvalidateRect(hWnd, NULL, TRUE);
          UpdateWindow(hWnd);
          break;

     case WM_ENABLE:
          hDC = GetDC(hWnd);
          if (wParam)
```

```
        PaintGButton(hWnd, hDC);
      else
        DisabledGButton(hWnd, hDC);
      ReleaseDC(hWnd, hDC);
      break;

case WM_SIZE:
      SizeGButton(hWnd, lParam);
      break;

case WM_ERASEBKGND:
      PaintFrameGButton(hWnd, (HDC)wParam);
      lResult = TRUE;
      break;

case WM_KILLFOCUS:
      if (GetPrivate & HASFOCUS)
       {
        hDC = GetDC(hWnd);
        FocusGButton(hWnd, hDC);
        ReleaseDC(hWnd, hDC);
        SetPrivate(GetPrivate & ~HASFOCUS & ~MOUSEACTIVATE);
       }
      lResult = TRUE;
      break;

case WM_MOUSEACTIVATE:
      if ((lResult = DefWindowProc(hWnd, wMsg, wParam, lParam)) == MA_ACTIVATE)
        SetPrivate(GetPrivate | MOUSEACTIVATE);
      break;

case WM_LBUTTONDBLCLK:
case WM_LBUTTONDOWN:
      if (GetPrivate & MOUSEACTIVATE && hWnd != GetFocus())
        SetFocus(hWnd);
      if (!(GetPrivate & MOUSEDOWN))
       {
        if ( !IsTypeButton(hWnd, DLGC_RADIOBUTTON) ||
            (!(GetPrivate & SELECTED) && IsTypeButton(hWnd, DLGC_RADIOBUTTON)))
         {
          hDC = GetDC(hWnd);
          FocusGButton(hWnd, hDC);
          if (GetPrivate & SELECTED)
            SetPrivate(GetPrivate & ~SELECTED | MOUSEDOWN);
          else
            SetPrivate(GetPrivate | MOUSEDOWN | SELECTED);
          MoveGButton(hWnd);
          BorderGButton(hWnd, hDC);
          FocusGButton(hWnd, hDC);
          ReleaseDC(hWnd, hDC);
         }
       }
      lResult = TRUE;
      break;

case WM_SETFOCUS:
      if (!(GetPrivate & HASFOCUS))
       {
        SetPrivate(GetPrivate | HASFOCUS);
        hDC = GetDC(hWnd);
        FocusGButton(hWnd, hDC);
        ReleaseDC(hWnd, hDC);
       }
      if (IsTypeButton(hWnd, DLGC_RADIOBUTTON) &&
```

```
          !(GetPrivate & SELECTED) && !(GetPrivate & MOUSEACTIVATE))
        {
         hDC = GetDC(hWnd);
         FocusGButton(hWnd, hDC);
         SetStyle(GetStyle | WS_TABSTOP);
         SetPrivate(GetPrivate | SELECTED);
         MoveGButton(hWnd);
         BorderGButton(hWnd, hDC);
         FocusGButton(hWnd, hDC);
         ReleaseDC(hWnd, hDC);
         UpdateCheckInGroup(hWnd);
        }
       lResult = TRUE;
       break;

case WM_PAINT:
       BeginPaint(hWnd, &ps);
       BorderGButton(hWnd, ps.hdc);
       PaintGButton(hWnd, ps.hdc);
       if (GetStyle & WS_DISABLED)
         DisabledGButton(hWnd, ps.hdc);
       FocusGButton(hWnd, ps.hdc);
       EndPaint(hWnd, &ps);
       lResult = TRUE;
       break;

case WM_KEYDOWN:
       if (wParam == VK_SPACE && !(GetPrivate & MOUSEDOWN) && !(GetPrivate & KEYDOWN))
        {
         if (!IsTypeButton(hWnd, DLGC_RADIOBUTTON))
          {
           SetCapture(hWnd);
           hDC = GetDC(hWnd);
           FocusGButton(hWnd, hDC);
           if (GetPrivate & SELECTED)
             SetPrivate(GetPrivate & ~SELECTED | KEYDOWN);
           else
             SetPrivate(GetPrivate | SELECTED | KEYDOWN);
           MoveGButton(hWnd);
           BorderGButton(hWnd, hDC);
           FocusGButton(hWnd, hDC);
           ReleaseDC(hWnd, hDC);
          }
        }
       lResult = TRUE;
       break;

case WM_KEYUP:
       if (wParam == VK_SPACE && !(GetPrivate & MOUSEDOWN) && GetPrivate & KEYDOWN)
        {
         ReleaseCapture();
         if (IsTypeButton(hWnd, DLGC_DEFPUSHBUTTON) ||
             IsTypeButton(hWnd, DLGC_UNDEFPUSHBUTTON))
          {
           hDC = GetDC(hWnd);
           FocusGButton(hWnd, hDC);
           if (GetPrivate & SELECTED)
             SetPrivate(GetPrivate & ~SELECTED & ~KEYDOWN);
           else
             SetPrivate(GetPrivate & ~KEYDOWN | SELECTED);
           MoveGButton(hWnd);
           BorderGButton(hWnd, hDC);
           FocusGButton(hWnd, hDC);
           ReleaseDC(hWnd, hDC);
```

```
            }
           else
             SetPrivate(GetPrivate & ~KEYDOWN);
           SendMessage(GetParent(hWnd), WM_COMMAND, GetWindowWord(hWnd, GWW_ID),
                       MAKELONG(hWnd, GBN_CLICKED));
          }
         lResult = TRUE;
         break;

    case WM_MOUSEMOVE:
         GetClientRect(hWnd, &rect);
         if (GetPrivate & KEYDOWN)
           break;
         if (PtInRect(&rect, MAKEPOINT(lParam)))
           break;
         ReleaseCapture();
         hDC = GetDC(hWnd);
         FocusGButton(hWnd, hDC);
         if (GetPrivate & SELECTED)
           SetPrivate(GetPrivate & ~SELECTED & ~MOUSEDOWN);
         else
           SetPrivate(GetPrivate & ~MOUSEDOWN | SELECTED);
         MoveGButton(hWnd);
         BorderGButton(hWnd, hDC);
         FocusGButton(hWnd, hDC);
         ReleaseDC(hWnd, hDC);
         lResult = TRUE;
         break;

     case WM_LBUTTONUP:
          if (GetPrivate & MOUSEDOWN)
           {
            ReleaseCapture();
            SetPrivate(GetPrivate & ~MOUSEDOWN);
            if (IsTypeButton(hWnd, DLGC_DEFPUSHBUTTON) ||
                IsTypeButton(hWnd, DLGC_UNDEFPUSHBUTTON))
             {
              hDC = GetDC(hWnd);
              FocusGButton(hWnd, hDC);
              if (GetPrivate & SELECTED)
                SetPrivate(GetPrivate & ~SELECTED);
              else
                SetPrivate(GetPrivate | SELECTED);
              MoveGButton(hWnd);
              BorderGButton(hWnd, hDC);
              FocusGButton(hWnd, hDC);
              ReleaseDC(hWnd, hDC);
             }
            else
             {
              if (IsTypeButton(hWnd, DLGC_RADIOBUTTON) && (GetPrivate & MOUSEACTIVATE))
               {
                UpdateCheckInGroup(hWnd);
                SetPrivate(GetPrivate & ~MOUSEACTIVATE);
                SetStyle(GetStyle | WS_TABSTOP);
               }
             }
            SendMessage(GetParent(hWnd), WM_COMMAND, GetWindowWord(hWnd, GWW_ID),
                        MAKELONG(hWnd, GBN_CLICKED));
           }
          lResult = TRUE;
          break;

     case WM_DESTROY:
```

```
            if (GetHBitmap)
              DeleteObject(GetHBitmap);
            break;

        case GBM_SETCHECK:
            if ((wParam && GetPrivate & SELECTED) ||
                (!wParam && !(GetPrivate & SELECTED)))
              break;
            hDC = GetDC(hWnd);
            FocusGButton(hWnd, hDC);
            if (!(GetPrivate & SELECTED))
             {
              SetStyle(GetStyle | WS_TABSTOP);
              SetPrivate(GetPrivate | SELECTED);
              UpdateCheckInGroup(hWnd);
             }
            else
             {
              SetStyle(GetStyle & ~WS_TABSTOP);
              SetPrivate(GetPrivate & ~SELECTED);
             }
            MoveGButton(hWnd );
            BorderGButton(hWnd, hDC);
            FocusGButton(hWnd, hDC);
            ReleaseDC(hWnd, hDC);
            break;

        case GBM_GETCHECK:
            if (GetPrivate & SELECTED)
              lResult = 1L;
            else
              lResult = 0L;
            break;

        case GBM_SETSTYLE:
            SetStyle(GetStyle & 0xFFFFFFE0 | (LONG)wParam);
            if (lParam)
             {
              InvalidateRect(hWnd, NULL, TRUE);
              UpdateWindow(hWnd);
             }
            lResult = TRUE;
            break;

        default:
            lResult = DefWindowProc(hWnd, wMsg, wParam, lParam);
            break;
      }
    return(lResult);
}
```

```
/*****************************************************************************************
  MODULE: Control.C

  PURPOSE: Contain the functions (generic) to interface object with dlg box
           editor.

  FUNCTIONS:

  ControlInfo()    - Allocate memory and stock info into structure
  AddControlType() - Add into dlg editor the control type of object
  ShowStyleDlg()   - Display in dlg editor the dialog box to select
                     style of object
  CtlStyleLock()   - Lock memory block which contain styles
  CtlStyleUnlock() - Unlock memory block which contain styles
  GetIdString()    - Call spcifique functions to convert string to an ID
  SetIdValue()     -  "      "          "


*****************************************************************************************/

#include <windows.h>
#include <custcntl.h>

#include "control.h"

/********** Definition of structures & constants ********************************/


static char _szCtlProp[] = "CtlDlgStyleData";

typedef struct {
                GLOBALHANDLE hCtlStyle;
                LPFNSTRTOID  lpfnStrToId;
                LPFNIDTOSTR  lpfnIdToStr;
               } CTLSTYLEDLG, FAR *LPCTLSTYLEDLG, NEAR *NPCTLSTYLEDLG;



GLOBALHANDLE FAR PASCAL ControlInfo (WORD wVersion, LPSTR szClass, LPSTR szTitle)
 {
  GLOBALHANDLE hMem = NULL;
  LPCTLINFO    lpCtlInfo;

  hMem = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, (DWORD) sizeof(CTLINFO));
  if (hMem == NULL)
    return(hMem);
  lpCtlInfo = (LPCTLINFO) GlobalLock(hMem);
  lpCtlInfo->wVersion  = wVersion;

  lpCtlInfo->wCtlTypes = 0;
  lstrcpy(lpCtlInfo->szClass, szClass);
  lstrcpy(lpCtlInfo->szTitle, szTitle);
  GlobalUnlock(hMem);
  return(hMem);
 }

BOOL FAR PASCAL AddControlType (GLOBALHANDLE hMem, WORD wType, WORD wWidth,
                                WORD wHeight, DWORD dwStyle, LPSTR szDescr)
 {
  LPCTLINFO lpCtlInfo; WORD wNumTypes;
  lpCtlInfo = (LPCTLINFO) GlobalLock(hMem);
  wNumTypes = lpCtlInfo->wCtlTypes;
  if (wNumTypes == CTLTYPES)
   {
```

```
    GlobalUnlock(hMem);
    return(FALSE);
   }
  lpCtlInfo->Type[wNumTypes].wType   = wType;
  lpCtlInfo->Type[wNumTypes].wWidth  = wWidth;
  lpCtlInfo->Type[wNumTypes].wHeight = wHeight;
  lpCtlInfo->Type[wNumTypes].dwStyle = dwStyle;
  lstrcpy(lpCtlInfo->Type[wNumTypes].szDescr, szDescr);
  lpCtlInfo->wCtlTypes++;
  GlobalUnlock(hMem);
  return(TRUE);
 }

int FAR PASCAL ShowStyleDlg (HANDLE hInstance, LPSTR szTemplate,
                             HWND hWndParent, FARPROC fpDlgProc, LONG lParam,
                             GLOBALHANDLE hCtlStyle, LPFNSTRTOID lpfnStrToId,
                             LPFNIDTOSTR lpfnIdToStr)
 {
  LOCALHANDLE hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  int x;

  hCtlStyleDlg = LocalAlloc(LMEM_MOVEABLE | LMEM_ZEROINIT, sizeof(CTLSTYLEDLG));
  if (hCtlStyleDlg == NULL)
    return(FALSE);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  npCtlStyleDlg->hCtlStyle = hCtlStyle;
  npCtlStyleDlg->lpfnStrToId = lpfnStrToId;
  npCtlStyleDlg->lpfnIdToStr = lpfnIdToStr;
  LocalUnlock(hCtlStyleDlg);
  SetProp(hWndParent, _szCtlProp, hCtlStyleDlg);
  x = DialogBoxParam(hInstance, szTemplate, hWndParent, fpDlgProc, lParam);
  RemoveProp(hWndParent, _szCtlProp);
  LocalFree(hCtlStyleDlg);
  return(x == IDOK);
 }

LPCTLSTYLE FAR PASCAL CtlStyleLock (HWND hDlg)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle = NULL;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(lpCtlStyle);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  lpCtlStyle = (LPCTLSTYLE) GlobalLock(npCtlStyleDlg->hCtlStyle);
  LocalUnlock(hCtlStyleDlg);
  return(lpCtlStyle);
 }

BOOL FAR PASCAL CtlStyleUnlock (HWND hDlg)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  BOOL          fOk = FALSE;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(fOk);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  fOk = GlobalUnlock(npCtlStyleDlg->hCtlStyle);
  LocalUnlock(hCtlStyleDlg);
```

```
  return(fOk);
 }

WORD FAR PASCAL GetIdString (HWND hDlg, LPSTR szId, WORD wIdMaxLen)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle;
  WORD          wIdLen;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(0);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  lpCtlStyle = (LPCTLSTYLE) GlobalLock(npCtlStyleDlg->hCtlStyle);
  wIdLen = (*npCtlStyleDlg->lpfnIdToStr)(lpCtlStyle->wId, szId, wIdMaxLen);
  GlobalUnlock(npCtlStyleDlg->hCtlStyle);
  LocalUnlock(hCtlStyleDlg);
  return(wIdLen);
 }

DWORD FAR PASCAL SetIdValue (HWND hDlg, LPSTR szId)
 {
  LOCALHANDLE   hCtlStyleDlg;
  NPCTLSTYLEDLG npCtlStyleDlg;
  LPCTLSTYLE    lpCtlStyle;
  DWORD         dwResult = 0;

  hCtlStyleDlg = GetProp(GetParent(hDlg), _szCtlProp);
  if (hCtlStyleDlg == NULL)
    return(dwResult);
  npCtlStyleDlg = (NPCTLSTYLEDLG) LocalLock(hCtlStyleDlg);
  dwResult = (*npCtlStyleDlg->lpfnStrToId)(szId);
  LocalUnlock(hCtlStyleDlg);
  if (LOWORD(dwResult) == 0)
    return(dwResult);
  lpCtlStyle = CtlStyleLock(hDlg);
  lpCtlStyle->wId = HIWORD(dwResult);
  CtlStyleUnlock(hDlg);
  return(dwResult);
 }
```

```
/*****************************************************************************
  MODULE: GButDlg.C

  PURPOSE: This file contain the functions to integrate the graphic button
           object into dialog boxes tools generator.

  FUNCTIONS:

  GraphicButtonInfo()  - Return to Windows the name of class,
                         the version and the title of object
  GraphicButtonStyle() - Is called by dialog box tools for used in this
                         the dialog box of specialization
  CheckSize()          - Check the corect size for the graphic button
  GraphicButtonDlgFn() - Function to process the dlg messages
  GraphicButtonFlags() - Build string to integrate in resources

*****************************************************************************/

#include <windows.h>
#include <custcntl.h>

#include "control.h"

#include "gbutton.h"
#include "private.h"
#include "dialog.h"

/********* Definition of constants ***************************************/

extern HANDLE _hInstance;
extern char   _szControlName[];

#ifdef DIALOG_EDITOR_30
extern DWORD  _dwStyle;
#endif

/********* Prototyping ***************************************************/

BOOL FAR PASCAL GraphicButtonDlgFn (HWND hDlg, WORD wMessage, WORD wParam, LONG lParam);



GLOBALHANDLE FAR PASCAL GraphicButtonInfo (void)
 {
  GLOBALHANDLE hCtlInfo = NULL;

  hCtlInfo = ControlInfo(0x0100, _szControlName, "None");
  if (hCtlInfo == NULL)
    return(hCtlInfo);
  AddControlType(hCtlInfo, 0, 24, 24, WS_BORDER | WS_CHILD | WS_TABSTOP | GBS_PUSHBUTTON | GBS_CLIP | GBS_ORNOT,
  return(hCtlInfo);
 }

BOOL FAR PASCAL GraphicButtonStyle (HWND hWnd, GLOBALHANDLE hCtlStyle,
                        LPFNSTRTOID lpfnStrToId, LPFNIDTOSTR lpfnIdToStr)
 {
  return(ShowStyleDlg(_hInstance, "STYLEDLG", hWnd, (FARPROC) GraphicButtonDlgFn,
                      0, hCtlStyle, lpfnStrToId, lpfnIdToStr));
 }

BOOL NEAR PASCAL CheckSize(LPSTR szTitle, WORD wCx, WORD wCy)
 {
  HANDLE           hFile;
  BITMAPFILEHEADER bitmapFileHeader;
```

```
  BITMAPINFOHEADER bitmapInfoHeader;
  OFSTRUCT         ofStruct;

  lstrcat((LPSTR)szTitle, (LPSTR)".BMP");
  if ((hFile = OpenFile( szTitle, &ofStruct, OF_READ)) == -1)
    return FALSE;
  if (_lread(hFile, (LPSTR)&bitmapFileHeader, sizeof(BITMAPFILEHEADER)) == -1)
   {
    _lclose(hFile);
    return FALSE;
   }
  if (_lread(hFile, (LPSTR)&bitmapInfoHeader, sizeof(BITMAPINFOHEADER)) == -1)
   {
    _lclose(hFile);
    return FALSE;
   }
  _lclose(hFile);
  if (bitmapInfoHeader.biWidth+(2*POSBITMAP) != (DWORD)wCy || bitmapInfoHeader.biHeight+(2*POSBITMAP) != (DWORD)w
    return FALSE;
  else
    return TRUE;
 }


BOOL FAR PASCAL GraphicButtonDlgFn (HWND hDlg, WORD wMsg, WORD wParam, LONG lParam)
 {
  BOOL       fResult = TRUE;
  char       szId[20];
  char       szTitle[CTLTITLE];
  DWORD      dwResult;
  WORD       wIndex;
  LPCTLSTYLE lpCtlStyle;
  OFSTRUCT   ofStruct;

  switch (wMsg) {
     case WM_INITDIALOG:
          // Init IDD_VALUE
          GetIdString(hDlg, szId, sizeof(szId));
          SetDlgItemText(hDlg, IDD_VALUE, (LPSTR)szId);
          // Init IDD_DIR
          OpenFile("", &ofStruct, OF_PARSE);
          SetDlgItemText(hDlg, IDD_DIR, (LPSTR)ofStruct.szPathName);
          // Init IDD_FILE
          SendDlgItemMessage(hDlg, IDD_FILE, CB_ADDSTRING, 0, (LONG)(LPSTR)"None");
          SendDlgItemMessage(hDlg, IDD_FILE, CB_DIR, 0x0000, (LONG)(LPSTR)"*.BMP");
          // Init current selection in IDD_FILE
          lpCtlStyle = CtlStyleLock(hDlg);
          if (!lstrlen((LPSTR)lpCtlStyle->szTitle) ||
              SendDlgItemMessage(hDlg, IDD_FILE, CB_SELECTSTRING, -1, (LONG)(LPSTR)lpCtlStyle->szTitle) == CB_ERR
           {
            SendDlgItemMessage(hDlg, IDD_FILE, CB_SELECTSTRING, -1, (LONG)(LPSTR)"None");
            EnableWindow(GetDlgItem(hDlg, IDD_RESIZE), FALSE);
            EnableWindow(GetDlgItem(hDlg, IDD_STRETCHED), FALSE);
           }
          if (!(lpCtlStyle->dwStyle & GBS_DEFPUSHBUTTON) &&
              !(lpCtlStyle->dwStyle & GBS_RADIOBUTTON) &&
              !(lpCtlStyle->dwStyle & GBS_AUTORADIOBUTTON) &&
              !(lpCtlStyle->dwStyle & GBS_CHECKBOX) &&
              !(lpCtlStyle->dwStyle & GBS_AUTOCHECKBOX))
            SendDlgItemMessage(hDlg, IDD_PUSH, BM_SETCHECK, TRUE, 0L);
          SendDlgItemMessage(hDlg, IDD_DEFPUSH, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_DEFPUSHBUTTON), 0L)
          SendDlgItemMessage(hDlg, IDD_RADIO, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_RADIOBUTTON), 0L);
          SendDlgItemMessage(hDlg, IDD_AUTORADIO, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_AUTORADIOBUTTON),
          SendDlgItemMessage(hDlg, IDD_CHECK, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_CHECKBOX), 0L);
```

```
        SendDlgItemMessage(hDlg, IDD_AUTOCHECK, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_AUTOCHECKBOX), 0L
        SendDlgItemMessage(hDlg, IDD_CLIP, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_CLIP), 0L);
        SendDlgItemMessage(hDlg, IDD_RESIZE, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_RESIZE), 0L);
        if ((BOOL)(lpCtlStyle->dwStyle & GBS_RESIZE) &&
            !CheckSize((LPSTR)lpCtlStyle->szTitle, lpCtlStyle->wCx, lpCtlStyle->wCy))
          SendDlgItemMessage(hDlg, IDD_CLIP, BM_SETCHECK, TRUE, 0L);
        SendDlgItemMessage(hDlg, IDD_STRETCHED, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_STRETCH), 0L);
        SendDlgItemMessage(hDlg, IDD_OR, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_OR), 0L);
        SendDlgItemMessage(hDlg, IDD_ORNOT, BM_SETCHECK, (BOOL)(lpCtlStyle->dwStyle & GBS_ORNOT), 0L);
        SendDlgItemMessage(hDlg, IDD_TABSTOP, BM_SETCHECK,
                           ((lpCtlStyle->dwStyle & WS_TABSTOP)?1:0), 0L);
        SendDlgItemMessage(hDlg, IDD_GROUP, BM_SETCHECK,
                           ((lpCtlStyle->dwStyle & WS_GROUP)?1:0), 0L);
        SendDlgItemMessage(hDlg, IDD_DISABLED, BM_SETCHECK,
                           ((lpCtlStyle->dwStyle & WS_DISABLED) > 0 ? 1 : 0), 0L);
        CtlStyleUnlock(hDlg);
        break;

    case WM_COMMAND:
        switch (wParam) {
          case IDOK:
            GetDlgItemText(hDlg, IDD_VALUE, szId, sizeof(szId));
            dwResult = SetIdValue(hDlg, szId);
            if (LOWORD(dwResult) == 0) break;
            wIndex = (WORD)SendDlgItemMessage(hDlg, IDD_FILE, CB_GETCURSEL, 0, 0L);
            SendDlgItemMessage(hDlg, IDD_FILE, CB_GETLBTEXT, wIndex, (LONG)(LPSTR)szTitle);
            lpCtlStyle = CtlStyleLock(hDlg);
            if (lstrcmp((LPSTR)szTitle, (LPSTR)"None") == 0)
              lstrcpy(lpCtlStyle->szTitle, "");
            else
             {
              szTitle[lstrlen((LPSTR)szTitle)-4] = '\0';
              lstrcpy(lpCtlStyle->szTitle, (LPSTR)szTitle);
             }
            lpCtlStyle->dwStyle &= 0xffff0000L;
            lpCtlStyle->dwStyle &= ~WS_TABSTOP;
            lpCtlStyle->dwStyle &= ~WS_GROUP;
            lpCtlStyle->dwStyle &= ~WS_DISABLED;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_TABSTOP, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= WS_TABSTOP;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_GROUP, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= WS_GROUP;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_DISABLED, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= WS_DISABLED;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_PUSH, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_PUSHBUTTON;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_DEFPUSH, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_DEFPUSHBUTTON;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_RADIO, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_RADIOBUTTON;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_AUTORADIO, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_AUTORADIOBUTTON;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_CHECK, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_CHECKBOX;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_AUTOCHECK, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_AUTOCHECKBOX;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_CLIP, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_CLIP;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_RESIZE, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_RESIZE;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_STRETCHED, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_STRETCH;
            if ((BOOL)SendDlgItemMessage(hDlg, IDD_OR, BM_GETCHECK, 0, 0L))
              lpCtlStyle->dwStyle |= GBS_OR;
```

```
                if ((BOOL)SendDlgItemMessage(hDlg, IDD_ORNOT, BM_GETCHECK, 0, 0L))
                  lpCtlStyle->dwStyle |= GBS_ORNOT;
#ifdef DIALOG_EDITOR_30
                _dwStyle = lpCtlStyle->dwStyle;
#endif
                CtlStyleUnlock(hDlg);

           case IDCANCEL:
                EndDialog(hDlg, wParam);
                break;

           case IDD_VALUE:
                if (HIWORD(lParam) == EN_CHANGE)
                 {
                  EnableWindow(GetDlgItem(hDlg, IDOK),
                        SendMessage(LOWORD(lParam), WM_GETTEXTLENGTH, 0, 0L)
                        ? TRUE : FALSE);
                 }
                break;

           case IDD_FILE:
                if (HIWORD(lParam) == CBN_SELCHANGE)
                 {
                  wIndex = (WORD)SendMessage(LOWORD(lParam), CB_GETCURSEL, 0, 0L);
                  SendMessage(LOWORD(lParam), CB_GETLBTEXT, wIndex, (LONG)(LPSTR)szTitle);
                  if (!lstrcmp((LPSTR)szTitle, (LPSTR)"None"))
                   {
                    SendDlgItemMessage(hDlg, IDD_RESIZE, BM_SETCHECK, FALSE, 0L);
                    SendDlgItemMessage(hDlg, IDD_STRETCHED, BM_SETCHECK, FALSE, 0L);
                    SendDlgItemMessage(hDlg, IDD_CLIP, BM_SETCHECK, TRUE, 0L);
                    EnableWindow(GetDlgItem(hDlg, IDD_RESIZE), FALSE);
                    EnableWindow(GetDlgItem(hDlg, IDD_STRETCHED), FALSE);
                   }
                  else
                   {
                    EnableWindow(GetDlgItem(hDlg, IDD_RESIZE), TRUE);
                    EnableWindow(GetDlgItem(hDlg, IDD_STRETCHED), TRUE);
                   }
                 }

           default: fResult = FALSE;
                    break;
         }
         break;

      default: fResult = FALSE;
               break;
   }
  return(fResult);
 }

WORD FAR PASCAL GraphicButtonFlags (DWORD dwFlags, LPSTR szString, WORD wMaxString)
 {
  WORD x;
  *szString = 0;

  if (!(dwFlags & GBS_DEFPUSHBUTTON) &&
      !(dwFlags & GBS_RADIOBUTTON) &&
      !(dwFlags & GBS_AUTORADIOBUTTON) &&
      !(dwFlags & GBS_CHECKBOX) &&
      !(dwFlags & GBS_AUTOCHECKBOX))
    lstrcat(szString, (LPSTR)"GBS_PUSHBUTTON | ");
  if (dwFlags & GBS_DEFPUSHBUTTON)
    lstrcat(szString, (LPSTR)"GBS_DEFPUSHBUTTON | ");
```

```
 if (dwFlags & GBS_RADIOBUTTON)
   lstrcat(szString, (LPSTR)"GBS_RADIOBUTTON | ");
 if (dwFlags & GBS_AUTORADIOBUTTON)
   lstrcat(szString, (LPSTR)"GBS_AUTORADIOBUTTON | ");
 if (dwFlags & GBS_CHECKBOX)
   lstrcat(szString, (LPSTR)"GBS_CHECKBOX | ");
 if (dwFlags & GBS_AUTOCHECKBOX)
   lstrcat(szString, (LPSTR)"GBS_AUTOCHECKBOX | ");
 if (dwFlags & GBS_CLIP)
   lstrcat(szString, (LPSTR)"GBS_CLIP | ");
 if (dwFlags & GBS_RESIZE)
   lstrcat(szString, (LPSTR)"GBS_RESIZE | ");
 if (dwFlags & GBS_STRETCH)
   lstrcat(szString, (LPSTR)"GBS_STRETCH | ");
 if (dwFlags & GBS_OR)
   lstrcat(szString, (LPSTR)"GBS_OR | ");
 if (dwFlags & GBS_ORNOT)
   lstrcat(szString, (LPSTR)"GBS_ORNOT | ");
 x = lstrlen(szString);
 if (x > 0) { x -= sizeof(" | ") - 1; *(szString + x) = 0; }
 return(x);
}
```

```
/*****************************************************************************
File name: GButton.H
*****************************************************************************/

#define CTLCOLOR_GBUTTON  (30)

// Graphic button control's class-specific window styles.

#define GBS_PUSHBUTTON      0x0000L
#define GBS_DEFPUSHBUTTON   0x0001L
#define GBS_CHECKBOX        0x0002L
#define GBS_AUTOCHECKBOX    0x0004L
#define GBS_RADIOBUTTON     0x0008L
#define GBS_AUTORADIOBUTTON 0x0010L
#define GBS_CLIP            0x0040L
#define GBS_RESIZE          0x0080L
#define GBS_STRETCH         0x0100L
#define GBS_OR              0x0200L
#define GBS_ORNOT           0x0400L

// Graphic button control's class-specific window messages.

#define GBM_GETCHECK       (WM_USER+0)
#define GBM_SETCHECK       (WM_USER+1)
// #define GBM_GETSTATE    (WM_USER+2)
// #define GBM_SETSTATE    (WM_USER+3)
#define GBM_SETSTYLE       (WM_USER+4)

// Graphic button control's notification codes to send to parent.

#define GBN_CLICKED         0
// #define GBN_PAINT        1
// #define GBN_HILITE       2
// #define GBN_UNHILITE     3
// #define GBN_DISABLE      4
#define GBN_DOUBLECLICKED   5
```

```
/*****************************************************************************
File name: Control.H
*****************************************************************************/

GLOBALHANDLE FAR PASCAL ControlInfo (WORD wVersion, LPSTR szClass, LPSTR szTitle);
BOOL FAR PASCAL AddControlType (GLOBALHANDLE hMem, WORD wType, WORD wWidth,
                                WORD wHeight, DWORD dwStyle, LPSTR szDescr);
int FAR PASCAL ShowStyleDlg (HANDLE hInstance, LPSTR szTemplate,
                             HWND hWndParent, FARPROC fpDlgProc, LONG lParam,
                             GLOBALHANDLE hCtlStyle, LPFNSTRTOID lpfnStrToId,
                             LPFNIDTOSTR lpfnIdToStr);
LPCTLSTYLE FAR PASCAL CtlStyleLock (HWND hDlg);
BOOL FAR PASCAL CtlStyleUnlock (HWND hDlg);
WORD FAR PASCAL GetIdString (HWND hDlg, LPSTR szId, WORD wIdMaxLen);
DWORD FAR PASCAL SetIdValue (HWND hDlg, LPSTR szId);
```

```
/*****************************************************************************
File name: Dialog.H
*****************************************************************************/

#define IDD_VALUE       100
#define IDD_DIR         101
#define IDD_FILE        102
#define IDD_CLIP        103
#define IDD_RESIZE      104
#define IDD_STRETCHED   105
#define IDD_PUSH        106
#define IDD_DEFPUSH     107
#define IDD_RADIO       108
#define IDD_CHECK       109
#define IDD_AUTORADIO   110
#define IDD_AUTOCHECK   111
#define IDD_TABSTOP     112
#define IDD_GROUP       113
#define IDD_DISABLED    114
#define IDD_OR          115
#define IDD_ORNOT       116
#define IDOK              1
#define IDCANCEL          2
```

```
/*****************************************************************************
File name: Private.H
*****************************************************************************/

#define POSBITMAP   (8)

#define DIALOG_EDITOR_30
```

```
STYLEDLG DIALOG LOADONCALL MOVEABLE DISCARDABLE  6,0,237,178
STYLE WS_POPUP | WS_CAPTION
CAPTION "Graphic Button Style ..."
FONT 8,"Helv"
BEGIN
    CONTROL "&ID Value:",-1,"static",WS_CHILD | SS_LEFT,10,17,35,8
    CONTROL "",IDD_VALUE,"edit",WS_TABSTOP | WS_CHILD | WS_BORDER | ES_LEFT,50,15,63,12
    CONTROL "Bit&map:",-1,"static",WS_CHILD | SS_LEFT,130,16,36,9
    CONTROL "",IDD_FILE,"combobox",CBS_DROPDOWNLIST | WS_TABSTOP | WS_CHILD | WS_VSCROLL,169,15,58,46
    CONTROL "Directory:",-1,"static",WS_CHILD | SS_LEFT,10,30,36,9
    CONTROL "c:\\windows",IDD_DIR,"static",WS_CHILD | SS_LEFT,50,30,175,9
    CONTROL "Behavior",-1,"button",BS_GROUPBOX | WS_GROUP | WS_CHILD,10,44,102,84
    CONTROL "&Push Button",IDD_PUSH,"button",BS_AUTORADIOBUTTON | WS_GROUP | WS_TABSTOP | WS_CHILD,15,53,75,12
    CONTROL "&DefPush Button",IDD_DEFPUSH,"button",BS_AUTORADIOBUTTON | WS_CHILD,15,65,75,12
    CONTROL "&Radio Button",IDD_RADIO,"button",BS_AUTORADIOBUTTON | WS_CHILD,15,77,75,12
    CONTROL "&Autoradio Button",IDD_AUTORADIO,"button",BS_AUTORADIOBUTTON | WS_CHILD,15,89,75,12
    CONTROL "C&hecked Button",IDD_CHECK,"button",BS_AUTORADIOBUTTON | WS_CHILD,15,101,75,12
    CONTROL "A&utoChecked Button",IDD_AUTOCHECK,"button",BS_AUTORADIOBUTTON | WS_CHILD,15,113,88,12
    CONTROL "Format",-1,"button",BS_GROUPBOX | WS_GROUP | WS_CHILD,130,44,97,50
    CONTROL "Bi&tmap Clipped",IDD_CLIP,"button",BS_AUTORADIOBUTTON | WS_GROUP | WS_TABSTOP | WS_CHILD,135,54,81,1
    CONTROL "&Window Resized",IDD_RESIZE,"button",BS_AUTORADIOBUTTON | WS_CHILD,135,66,85,12
    CONTROL "Bitmap &Stretched",IDD_STRETCHED,"button",BS_AUTORADIOBUTTON | WS_CHILD,135,79,81,12
    CONTROL "Ta&bstop",IDD_TABSTOP,"BUTTON",BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_CHILD,95,137,42,12
    CONTROL "&Group",IDD_GROUP,"BUTTON",BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_CHILD,152,137,38,12
    CONTROL "&Ok",IDOK,"button",WS_GROUP | WS_TABSTOP | WS_CHILD | BS_PUSHBUTTON,67,156,42,14
    CONTROL "&Cancel",IDCANCEL,"button",WS_GROUP | WS_TABSTOP | WS_CHILD | BS_PUSHBUTTON,126,156,42,14
    CONTROL "Disab&led",IDD_DISABLED,"button",BS_AUTOCHECKBOX | WS_GROUP | WS_TABSTOP | WS_CHILD,39,137,41,12
    CONTROL "Or (&1)",IDD_OR,"button",BS_AUTORADIOBUTTON | WS_GROUP | WS_TABSTOP | WS_CHILD,135,101,36,12
    CONTROL "Or not (&2)",IDD_ORNOT,"button",BS_AUTORADIOBUTTON | WS_CHILD,135,113,41,12
    CONTROL "Grayed operation",-1,"button",BS_GROUPBOX | WS_GROUP | WS_CHILD,130,94,97,34
END
```

```
; Module name: GButton.DEF

LIBRARY     GBUTTON
DESCRIPTION 'Graphic Button Custom Control Library'
EXETYPE     WINDOWS
STUB        'WinStub.Exe'

CODE        MOVEABLE DISCARDABLE SHARED PRELOAD
DATA        MOVEABLE SINGLE PRELOAD

HEAPSIZE    1024

EXPORTS
   WEP                 @1    RESIDENTNAME
   GraphicButtonInfo   @2
   GraphicButtonStyle  @3
   GraphicButtonFlags  @4
   GraphicButtonWndFn  @5
   GraphicButtonDlgFn  @6
```

```
/*****************************************************************************
File name: GButton.RC
*****************************************************************************/

#include <windows.h>

#include "gbutton.h"
#include "dialog.h"
#include "gbutton.dlg"
```

```
#*****************************************************************************
#File name: MAKEFILE
#*****************************************************************************

PROG  = GBut
MODEL = S
CFLAGS = -A$(MODEL)w -D_WINDOWS -D_WINDLL -Gcsw2 -W4 -Zlepid -Od
LFLAGS = /NOE/BA/A:16/M/CO/LI/F
LIBS = $(MODEL)nocrtd + $(MODEL)dllcew + libw

M1 = $(PROG)ton.obj $(PROG)Dlg.obj Control.obj

ICONS   =
BITMAPS =
CURSORS =
RESOURCES = $(ICONS) $(BITMAPS) $(CURSORS)

.SUFFIXES: .rc

.rc.res:
    rc -r $*.rc

$(PROG)ton.DLL: $(M1) $(PROG)ton.Def $(PROG)ton.Res
     link $(LFLAGS) @<<$(PROG)ton.Lnk
C:\WINDEV\LIB\LibEntry.obj $(M1)
$(PROG)ton.DLL, $(PROG)ton, $(LIBS), $(PROG)ton
<<
     rc -Fe$(PROG)ton.DLL $(PROG)ton.Res
     copy $(PROG)ton.dll c:\win3

$(PROG)ton.obj:    $*.c $*.h private.h
$(PROG)Dlg.obj: $*.c $(PROG)ton.h dialog.h private.h
Control.obj:      $*.c $*.h

$(PROG)ton.res:   $*.rc $*.h $*.dlg dialog.h $(RESOURCES)
```

4. Objet interactif de type "StateLine".

```
/*****************************************************************************

   MODULE:   statline.c

   PURPOSE:  Make & manage one Status bar in a main Window.

   FUNCTIONS:

   StatLineWndProc()      - Process window messages.
   RegisterControlClass() - Register Status Line control Class
   SetTextPos()           - Place a text in positionned frame
   PutText()              - Draw the text
   DefineFrame()          - Define the positionned frames
   DrawFrame()            - Draw the positionned frames
   DrawWin()              -
   DrawWhiteLine()        - Draw the upper white line
   CopyFont()             - Copy the logical font defined by the user
   DisplayTime()          - Display clock in sl if activate

*****************************************************************************

   COMMENT : - The Window style must be have the WS_BORDER at CreateWindow.
             - If you want use a clock in status bar, this file must be
               compiled in LARGE model.

*****************************************************************************/

#include "windows.h"
#include "statline.h"
#include "stdio.h"
#include "time.h"

/****** Definition of strutures & constants ***********************************/

struct table
        {
         WORD   Pos;
         WORD   Length;
         HANDLE hText;
        } ;
typedef struct table DEFTAB;

HANDLE _hInstance       = NULL;
char   _szControlName[] = "StatusLine";

#define  SLFULL         (0x0001)
#define  SLCLOCK        (0x0002)
#define  BORDER         (1)
#define  HORIZ          (3)

/****** Definition of control extra bytes (words) *****************************/

#define  CBWNDEXTRA     (12)
#define  GWW_TABLE      (0)
#define  GWW_NUMBER     (2)
#define  GWW_WIDTH      (4)
#define  GWW_HFONT      (6)
#define  GWW_FLAGS      (8)
#define  GWW_POSCLK     (10)

/****** Prototyping ***********************************************************/

long FAR PASCAL StatLineWndProc(HWND hWnd, unsigned message, unsigned wParam, LONG lParam);
static BOOL NEAR PASCAL RegisterControlClass(HANDLE hInstance);
```

```
static BOOL NEAR PASCAL SetTextPos(HDC hDC,HWND hWnd,RECT Rect , WORD wParam);
static BOOL NEAR PASCAL DefineFrame(HWND hWnd, WORD wParam, LONG lParam);
static VOID NEAR PASCAL DrawFrame(HWND hWnd, HDC hDC,LPRECT lpRect);
static VOID PASCAL DrawWhiteLine(HDC hDC, LPRECT lpRect);
static VOID NEAR PASCAL DrawWin(HWND hWnd,HDC hDC,LPRECT lpRect,DEFTAB FAR *lpDefTab);
static VOID NEAR PASCAL CopyFont(HWND hWnd, HFONT hUserFont);
static VOID PASCAL PutText(HDC hDC,RECT lpRect,LPSTR szBuffer,int Alinea);
static VOID PASCAL DisplayTime(HWND hWnd);



BOOL FAR PASCAL LibMain (HANDLE hModule, WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdLine)
  {
   if (_hInstance == NULL)
    {
     if (wHeapSize != 0)
       UnlockData(0);
     _hInstance = (RegisterControlClass(hModule) ? hModule : NULL);
     }
   return(_hInstance ? TRUE : FALSE);
  }

VOID FAR PASCAL WEP (int nSystemExit)
 {
  return;
 }

static BOOL NEAR PASCAL RegisterControlClass(HANDLE hInstance)
{
  WNDCLASS   WndClass;

  WndClass.style         = CS_GLOBALCLASS ;
  WndClass.lpfnWndProc   = StatLineWndProc;
  WndClass.cbClsExtra    = 0;
  WndClass.cbWndExtra    = CBWNDEXTRA;
  WndClass.hInstance     = hInstance;
  WndClass.hIcon         = NULL;
  WndClass.hCursor       = LoadCursor(NULL, IDC_ARROW);
  WndClass.hbrBackground = (HBRUSH)COLOR_BTNFACE+1;
  WndClass.lpszMenuName  = NULL;
  WndClass.lpszClassName = (LPSTR)_szControlName;

  return (RegisterClass((LPWNDCLASS)&WndClass));
}

long FAR PASCAL StatLineWndProc(HWND hWnd, unsigned message, unsigned wParam, LONG lParam)
 {
  TEXTMETRIC     tm;
  PAINTSTRUCT    ps;
  HDC            hDC;
  RECT           rStat;
  HFONT          hFont;
  WORD           Flags,i,Number;
  int            cyStatLine;
  HANDLE         hDefTab,hBuffer;
  DEFTAB FAR *   lpDefTab;
  LPSTR          szBuffer;

    switch (message)
         {
         case WM_CREATE:
              hDC=GetDC(hWnd);
              GetTextMetrics(hDC,&tm);
              ReleaseDC(hWnd,hDC);
```

```
        cyStatLine=tm.tmHeight+ tm.tmExternalLeading+8;
        SetWindowWord(hWnd,GWW_WIDTH,(WORD)cyStatLine);
        SetWindowWord(hWnd,GWW_HFONT,0);
        return TRUE;
    case SL_ADJUST:
        cyStatLine=GetWindowWord(hWnd,GWW_WIDTH);
        MoveWindow(hWnd,
                   -GetSystemMetrics(SM_CXBORDER),
                   HIWORD(lParam)-cyStatLine+1,
                   LOWORD(lParam)+2*GetSystemMetrics(SM_CXBORDER),
                   cyStatLine,
                   TRUE);
        InvalidateRect(hWnd,NULL,TRUE);
        return TRUE;
    case SL_DEFINE :
        DefineFrame(hWnd,(WORD) wParam,(LONG) lParam);
        return TRUE;

    case WM_SETFONT:
        CopyFont(hWnd, wParam);
        hDC=GetDC(hWnd);
        hFont=SelectObject(hDC,(HFONT)GetWindowWord(hWnd,GWW_HFONT));
        GetTextMetrics(hDC,&tm);
        cyStatLine=tm.tmHeight+ tm.tmExternalLeading+8;
        SetWindowWord(hWnd,GWW_WIDTH,(WORD)cyStatLine);
        SelectObject(hDC,hFont);
        ReleaseDC(hWnd,hDC);
        GetClientRect(GetParent(hWnd),(LPRECT)&rStat);
        SendMessage(hWnd,SL_ADJUST,0,MAKELONG(rStat.right,rStat.bottom));
        return TRUE;
    case SL_SETTEXTPOS:
        hDC=GetDC(hWnd);
        GetClientRect(hWnd,(LPRECT)&rStat);
        if(wParam<GetWindowWord(hWnd,GWW_NUMBER))
         {
          hDefTab=GetWindowWord(hWnd,GWW_TABLE);
          lpDefTab=(DEFTAB FAR *)GlobalLock(hDefTab);
          if((UINT)lstrlen((LPSTR)lParam)>(UINT)(lpDefTab+wParam)->Length)
           *(LPSTR)(lParam+(lpDefTab+wParam)->Length-1)='\0';
          szBuffer=(LPSTR)GlobalLock((lpDefTab+wParam)->hText);
          lstrcpy((LPSTR)szBuffer,(LPSTR)lParam);
          GlobalUnlock((lpDefTab+wParam)->hText);
          GlobalUnlock(hDefTab);
          SetTextPos(hDC,hWnd,rStat,(WORD)wParam);
         }
        ReleaseDC(hWnd,hDC);
        return TRUE;
    case SL_SETCLOCK:
        if(!(GetWindowWord(hWnd,GWW_FLAGS) & SLCLOCK))
         {
          if(SetTimer(hWnd,1,(UINT)60000,NULL))
           {
            hDefTab=GetWindowWord(hWnd,GWW_TABLE);
            if(hDefTab)
             {
              SetWindowWord(hWnd,GWW_POSCLK,(WORD) wParam);
              Flags=GetWindowWord(hWnd,GWW_FLAGS);
              SetWindowWord(hWnd,GWW_FLAGS,(WORD)(Flags|SLCLOCK));
              lpDefTab=(DEFTAB FAR *)GlobalLock(hDefTab);
              (lpDefTab+wParam)->Length=8;
              GlobalUnlock(hDefTab);
             }
           }
         }
```

```
        return TRUE;
    case WM_TIMER:
        if(GetWindowWord(hWnd,GWW_FLAGS) & SLCLOCK )
         DisplayTime(hWnd);
        return TRUE;
    case WM_SETTEXT:
        Flags=GetWindowWord(hWnd,GWW_FLAGS);
        if(NULL==(LPSTR)lParam)
          SetWindowWord(hWnd,GWW_FLAGS,(WORD)(Flags&(~SLFULL)));
        else
          SetWindowWord(hWnd,GWW_FLAGS,(WORD)(Flags|SLFULL));
        DefWindowProc(hWnd, message, wParam, lParam);
        GetClientRect(hWnd,(LPRECT)&rStat);
        rStat.top+=2*BORDER;
        InvalidateRect(hWnd,(LPRECT)&rStat,TRUE);
        return TRUE;
    case WM_PAINT:
        hFont=0;
        GetClientRect(hWnd,(LPRECT)&rStat);
        hDC=BeginPaint(hWnd,(LPPAINTSTRUCT)&ps);
        if(!(GetWindowWord(hWnd,GWW_FLAGS) & (SLFULL)) )
         {
          if(GetWindowWord(hWnd,GWW_TABLE))
           {
            DrawFrame(hWnd, hDC, (LPRECT)&rStat);
            Number=GetWindowWord(hWnd,GWW_NUMBER);
            for(i=0;i<Number;i++)
             {
              SetTextPos(hDC,hWnd,rStat,i);
             }
           }
          if(GetWindowWord(hWnd,GWW_FLAGS) & SLCLOCK)
           {
            DisplayTime(hWnd);
           }
         }
        else
         {
          i=LOWORD(SendMessage(hWnd,WM_GETTEXTLENGTH,0,0L));
          hBuffer=GlobalAlloc(GHND,(i+1)*sizeof(char));
          szBuffer=GlobalLock(hBuffer);
          SendMessage(hWnd,WM_GETTEXT,(WORD)i+1,(LONG)(LPSTR)szBuffer);
          if(GetWindowWord(hWnd,GWW_HFONT))
            hFont=SelectObject(hDC,(HFONT)GetWindowWord(hWnd,GWW_HFONT));
          rStat.top+=2*BORDER;
          PutText(hDC,rStat,(LPSTR)szBuffer, 2);
          if(hFont) SelectObject(hDC,hFont);
          GlobalUnlock(hBuffer);
          GlobalFree(hBuffer);
         }
        EndPaint(hWnd,(LPPAINTSTRUCT)&ps);
        return TRUE;

    case WM_DESTROY:
        hDefTab=GetWindowWord(hWnd,GWW_TABLE);
        if(hDefTab)
         {
          Number=GetWindowWord(hWnd,GWW_NUMBER);
          lpDefTab=(DEFTAB FAR *)GlobalLock(hDefTab);
          for(i=0;i<Number;i++)
           {
            GlobalFree((lpDefTab+i)->hText);
           }
          GlobalUnlock(hDefTab);
```

```
                GlobalFree(hDefTab);
               }
               if(GetWindowWord(hWnd,GWW_HFONT))
                DeleteObject((HFONT)GetWindowWord(hWnd,GWW_HFONT));
               if(GetWindowWord(hWnd,GWW_FLAGS) & SLCLOCK)
                KillTimer(hWnd,1);

              return TRUE;

        }
  return (DefWindowProc(hWnd, message, wParam, lParam));
 }


static BOOL NEAR PASCAL SetTextPos(HDC hDC, HWND hWnd, RECT Rect, WORD Pos)
 {
  TEXTMETRIC  tm;
  HANDLE      hDefTab;
  HFONT       hFont=0;
  RECT        rec;
  DEFTAB FAR *lpDefTab;
  int         xChar,yChar;
  LPSTR       szBuffer;

  hDefTab=GetWindowWord(hWnd,GWW_TABLE);
  if(hDefTab)
   {
    lpDefTab=(DEFTAB FAR *)GlobalLock(hDefTab);
    if(!(lpDefTab+Pos)->hText)
     {
      GlobalUnlock(hDefTab);
      return FALSE;
     }
    if(GetWindowWord(hWnd,GWW_HFONT))
      hFont=SelectObject(hDC,(HFONT)GetWindowWord(hWnd,GWW_HFONT));
    GetTextMetrics(hDC,&tm);
    xChar = tm.tmAveCharWidth;
    yChar = tm.tmHeight + tm.tmExternalLeading;
    szBuffer=(LPSTR)GlobalLock((lpDefTab+Pos)->hText);
    SetRect(&rec,Rect.left+(lpDefTab+Pos)->Pos*xChar,
                 Rect.bottom-HORIZ-yChar,
                 Rect.left+(lpDefTab+Pos)->Pos*xChar+(lpDefTab+Pos)->Length*xChar,
                 Rect.bottom-HORIZ);
    PutText(hDC,rec,(LPSTR) szBuffer,1);
    GlobalUnlock((lpDefTab+Pos)->hText);
    GlobalUnlock(hDefTab);
    if(hFont) SelectObject(hDC,hFont);
   }
  else
   {
    return FALSE;
   }
  return TRUE;
 }

static VOID PASCAL PutText(HDC hDC,RECT Rect,LPSTR szBuffer,int Alinea)
 {
  LOGBRUSH    logBrush;
  HBRUSH      hBrush;
  TEXTMETRIC  tm;
  int         xChar;

   GetTextMetrics(hDC,&tm);
   xChar = tm.tmAveCharWidth;
```

```
    logBrush.lbStyle=BS_SOLID;
    logBrush.lbColor=GetSysColor(COLOR_BTNFACE);
    hBrush=CreateBrushIndirect((LPLOGBRUSH)&logBrush);
    FillRect(hDC,(LPRECT)&Rect,hBrush);
    DeleteObject(hBrush);
    SetBkMode(hDC,TRANSPARENT);
    Rect.left=Rect.left+xChar*Alinea;
    DrawText(hDC,(LPSTR)szBuffer,lstrlen((LPSTR)szBuffer),(LPRECT)&Rect,DT_LEFT|DT_SINGLELINE|DT_VCENTER);
  }

static BOOL NEAR PASCAL DefineFrame(HWND hWnd, WORD wParam, LONG lParam)
  {
   WORD          i;
   HANDLE        hDefTab;
   DEFTAB  FAR * lpDefTab;
   UDEFTAB FAR * lpUserDefTab;

   if(LOWORD(lParam)==0) return FALSE;
   hDefTab=GetWindowWord(hWnd,GWW_TABLE);
   if(hDefTab) GlobalFree(hDefTab);
   hDefTab=GlobalAlloc(GHND,wParam*sizeof(DEFTAB));
   lpDefTab=(DEFTAB FAR *)GlobalLock(hDefTab);
   lpUserDefTab=(UDEFTAB FAR *)GlobalLock(LOWORD(lParam));
   for(i=0;i<wParam;i++)
    {
     lpDefTab->Pos=lpUserDefTab->Pos;
     lpDefTab->Length=lpUserDefTab->Length;
     lpDefTab->hText=GlobalAlloc(GHND,lpDefTab->Length*sizeof(char));
     lpDefTab+=1;
     lpUserDefTab+=1;
    }
   GlobalUnlock(hDefTab);
   GlobalUnlock(LOWORD(lParam));
   SetWindowWord(hWnd,GWW_NUMBER,wParam);
   SetWindowWord(hWnd,GWW_TABLE,(WORD)hDefTab);
   InvalidateRect(hWnd,NULL,TRUE);
   return TRUE;
  }

static VOID NEAR PASCAL DrawFrame(HWND hWnd, HDC hDC,LPRECT lpRect)
  {
   int           i,Number;
   DEFTAB FAR * lpDefTab;

   DrawWhiteLine(hDC,lpRect);
   Number=GetWindowWord(hWnd,GWW_NUMBER);

   lpDefTab=(DEFTAB FAR *)GlobalLock(GetWindowWord(hWnd,GWW_TABLE));
   for(i=0;i<Number;i++)
    {
     DrawWin(hWnd,hDC,lpRect,lpDefTab);
     lpDefTab+=1;
    }
   GlobalUnlock(GetWindowWord(hWnd,GWW_TABLE));

  }
static VOID PASCAL DrawWhiteLine(HDC hDC, LPRECT lpRect)
   {
     HPEN hOldPen;

     hOldPen=SelectObject(hDC,GetStockObject(WHITE_PEN));
     MoveTo(hDC, lpRect->left, 0);
     LineTo(hDC, lpRect->right, 0);
```

```
    SelectObject(hDC,hOldPen);

   }
static VOID NEAR PASCAL DrawWin(HWND hWnd,HDC hDC,LPRECT lpRect,DEFTAB FAR *lpDefTab)
 {
  TEXTMETRIC   tm;
  HFONT        hFont=0;
  POINT        point[4];
  HPEN         hOldPen;
  int          xChar,yChar;

  if(GetWindowWord(hWnd,GWW_HFONT))
    hFont=SelectObject(hDC,(HFONT)GetWindowWord(hWnd,GWW_HFONT));
  GetTextMetrics(hDC,&tm);
  xChar = tm.tmAveCharWidth;
  yChar = tm.tmHeight + tm.tmExternalLeading;
  if(hFont) SelectObject(hDC,hFont);

    hOldPen=SelectObject(hDC,GetStockObject(WHITE_PEN));
    point[0].x=lpRect->left+lpDefTab->Pos*xChar-BORDER;
    point[0].y=lpRect->bottom-HORIZ+BORDER;
    point[1].x=lpRect->left+lpDefTab->Pos*xChar+lpDefTab->Length*xChar+BORDER;
    point[1].y=lpRect->bottom-HORIZ+BORDER;
    point[2].x=lpRect->left+lpDefTab->Pos*xChar+lpDefTab->Length*xChar+BORDER;
    point[2].y=lpRect->bottom-HORIZ-yChar-BORDER;
    Polyline(hDC,(LPPOINT)&point,3);
    hOldPen=SelectObject(hDC,CreatePen(PS_SOLID,0,RGB(128,128,128)));
    point[0].x=lpRect->left+lpDefTab->Pos*xChar+lpDefTab->Length*xChar+BORDER;
    point[0].y=lpRect->bottom-HORIZ-yChar-BORDER;
    point[1].x=lpRect->left+lpDefTab->Pos*xChar-BORDER;
    point[1].y=lpRect->bottom-HORIZ-yChar-BORDER;
    point[2].x=lpRect->left+lpDefTab->Pos*xChar-BORDER;
    point[2].y=lpRect->bottom-HORIZ+BORDER;
    Polyline(hDC,(LPPOINT)&point,3);

  DeleteObject(SelectObject(hDC,hOldPen));
 }

static VOID PASCAL DisplayTime(HWND hWnd)
 {
  time_t    lTime;
  struct tm *Time;
  char      szBuffer[25];

  time(&lTime);
  Time=localtime(&lTime);
  if(Time->tm_min<10)
   sprintf(szBuffer,"%2d:0%1d",Time->tm_hour,Time->tm_min);
  else
   sprintf(szBuffer,"%2d:%2d",Time->tm_hour,Time->tm_min);
  SendMessage(hWnd,SL_SETTEXTPOS,GetWindowWord(hWnd,GWW_POSCLK),(LONG)(LPSTR)szBuffer);

 }

static VOID NEAR PASCAL CopyFont(HWND hWnd, HFONT hUserFont)
 {
  LOGFONT    lf;
  HFONT      hFont;

  if(GetWindowWord(hWnd,GWW_HFONT))
    DeleteObject((HFONT)GetWindowWord(hWnd,GWW_HFONT));
  GetObject(hUserFont,sizeof(LOGFONT),(LPSTR)&lf);
  hFont=CreateFontIndirect((LPLOGFONT)&lf);
  SetWindowWord(hWnd,GWW_HFONT,(WORD)hFont);
```

```
long FAR PASCAL MainWndProc(HWND, unsigned, unsigned , LONG);
BOOL FAR PASCAL DlgProc( HWND hDlg, unsigned message, WORD wParam, LONG lParam);
BOOL InitClass(HANDLE hInstance) ;
BOOL InitMainInst(HANDLE hInstance, int nCmdShow) ;
void DoCaption(char *szTitle);
```

```
/*****************************************************************************
    FILE        : Statline.h

    DESCRIPTION : file header of Status Line control function

*****************************************************************************/
struct usertable
        {
         WORD   Pos;
         WORD   Length;
        } ;
typedef struct usertable UDEFTAB;


#define  CTLCOLOR_STATLINE (25)

#define  SL_ADJUST          (WM_USER+0)
#define  SL_SETTEXT         (WM_USER+1)
#define  SL_SETTEXTPOS      (WM_USER+2)
#define  SL_GETTEXT         (WM_USER+3)
#define  SL_DEFINE          (WM_USER+4)
#define  SL_SETCLOCK        (WM_USER+5)
```

```
; Module name: Statline.DEF

LIBRARY     STATUSLINE
DESCRIPTION 'Status Line Custom Control Library'
EXETYPE     WINDOWS
STUB        'WinStub.Exe'

; CODE        PRELOAD FIXED NONDISCARDABLE
; DATA        MOVEABLE PRELOAD SINGLE

CODE        MOVEABLE DISCARDABLE SHARED PRELOAD
DATA        MOVEABLE SINGLE PRELOAD

HEAPSIZE    1024

EXPORTS
   WEP                    @1 RESIDENTNAME
   StatLineWndProc        @5
```

```
/*****************************************************************************
File name: Statline.RC
*****************************************************************************/

#include <windows.h>

#include "statline.h"
```

```
#*****************************************************************************
#Module name: Statline.make
#Programmer : MTS
#*****************************************************************************

PROG  = Statline
MODEL = L
CFLAGS = -A$(MODEL)w -D_WINDOWS -D_WINDLL -Gcsw2 -W4 -Zlepid -Od
LFLAGS = /NOE/BA/A:16/M/CO/LI/F
LIBS = $(MODEL)dllcew + libw

M1 = $(PROG).obj

ICONS   =
BITMAPS =
CURSORS =
RESOURCES = $(ICONS) $(BITMAPS) $(CURSORS)

#.SUFFIXES: .rc

#.rc.res:
#    rc -r $*.rc

$(PROG).DLL: $(M1) $(PROG).Def
     link $(LFLAGS) @<<$(PROG).Lnk
C:\WINDEV\LIB\LibEntry.obj $(M1)
$(PROG).DLL, $(PROG), $(LIBS), $(PROG)
<<
#     rc -Fe$(PROG).DLL $(PROG).Res
     copy $(PROG).dll c:\windows

$(PROG).obj:    $*.c $*.h
#$(PROG)Dlg.obj: $*.c $(PROG).h  dialog.h
Cntl-DE.obj:    $*.c $*.h

#$(PROG).res:   $*.rc $*.h $*.dlg dialog.h $(RESOURCES)
```