MapBasic “Dialog”

ses “Controls”

et quelques énoncés reliés

Définitions

Conditions d’utilisation

et

Exemples de mise en œuvre

Jacques Paris

Octobre 2006

Mes remerciements vont à Maurice Nadal qui m'a donné l'idée de ce document, les encouragements pour le développer et l'aide critique dans sa mise en forme.

Jacques Paris

Montréal, octobre 2006

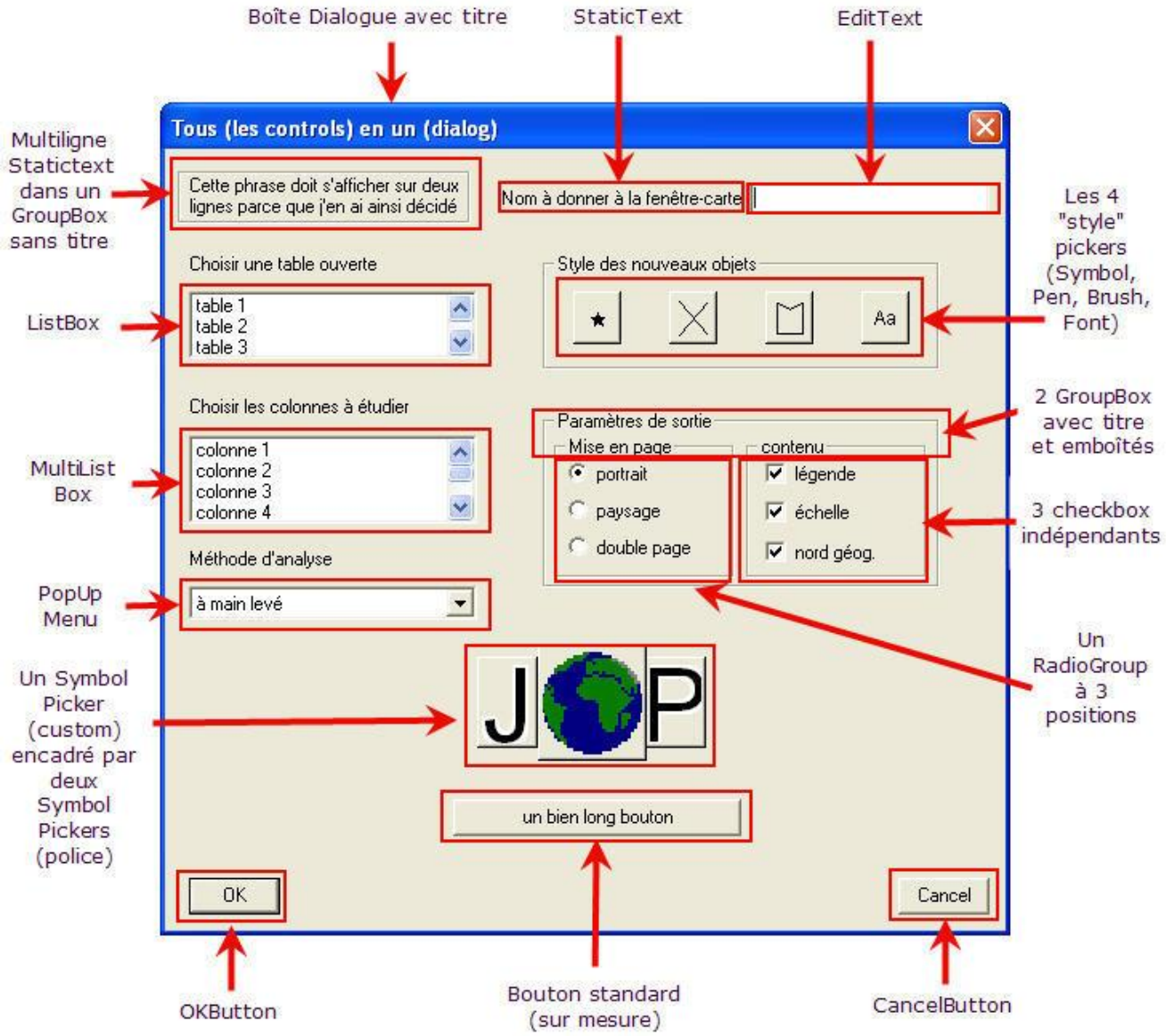
Table des matières

J'ai adopté l'orthographe anglaise quand il s'agit de mots de code MB et la française lorsque référence est faite à la « fenêtre » et à ses composantes. J'espère que j'ai été systématique en cela et m'excuse si j'ai eu quelques ratés

Tous en Un

Ce dialogue regroupe tous les contrôles qui ont existé dans MapBasic depuis la version 4. Ce domaine a probablement été un des plus stables de tout MapBasic. Ce n'est qu'avec la version 7 qu'est apparu un nouveau contrôle non inclus dans cette image, DocumentWindow.

> *J'ai donné libre cours à mon imagination durant la construction de ce dialogue qui n'a pas grand-chose à voir avec la réalité*



L'énoncé « DIALOG»

Cette commande doit être suivie par au moins un contrôle pour être reconnue par MapInfo et par n'importe quelle combinaison de mots-clés avec les données pertinentes.

MOTS CLÉS : Title, Height, Width, Position, Calling, Control

TITLE "*chaîne de caractères*"

La longueur du texte ne détermine pas la taille du dialogue. Il risque d'être tronqué parce que la largeur du dialogue, si elle est spécifiée n'est pas assez grande ou que la largeur du dialogue n'étant pas spécifiée son contenu ne définit pas une largeur suffisante.

Si Title n'est pas présent, il n'y a aucun cadre ni barre titre.

> *ATTENTION . Comme il n'a pas de barre de titre, un dialogue « sans titre » ne peut pas être déplacé (aucun endroit pour le « saisir ») et il ne peut pas être fermé par la petite croix du coin droit. Si le dialogue ne comporte pas de OkButton ni de CancelButton, ce dialogue ne pourra pas être fermé.*

```
dialog width 38 height 15
control okbutton position 0,0
```

Dialogue super minimaliste
Largeur et hauteur définies
par le contenu du dialogue

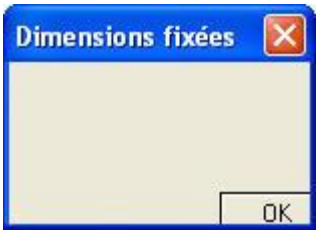
HEIGHT *h* & WIDTH *w*

La hauteur et la largeur exprimées en UD (Unités de Dialogue) fixent les dimensions de la boîte « grise » du dialogue (l'intérieur du cadre bleu de l'image suivante).

```
Dialog Title "Dimensions fixées" width 100 height 50
Control OKbutton
```

Hauteur et largeur spécifiées (l'une ou l'autre ou les 2)

```
Dialog Title "Dimensions fixées" width 100 height 50
    Control OKbutton position 70, 40
```



Une partie du contenu débordant du cadre fixé est ignorée

POSITION *X, Y*

La paire de valeurs X,Y exprimées en UD détermine la position du dialogue par rapport à la zone de travail de MapInfo. Le point de référence sur le dialogue est le coin supérieur gauche de la partie grise, celle qui est tout le dialogue quand il n'y a aucun cadre (voir plus haut sous TITLE). Le point de référence dans la fenêtre MapInfo dépend de la présence de la barre de menu.

```
Dialog title "Ok test" position 0,0
Control OKbutton
```

Avec la barre de menus Sans la barre de menus

La zone de travail de MI couvre donc l'intérieur « gris » de la fenêtre plus l'espace occupé par les barres d'outils.

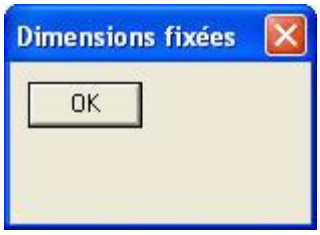
Si POSITION est omis, le centre du dialogue au complet et celui de la fenêtre MapInfo coïncident, sauf si par une combinaison de position de la fenêtre MI et de taille du dialogue le haut ou le bord gauche du dialogue« bute » contre les limites de l'écran même.

CALLING *Dialog_Handler*

Si l'énoncé Dialog fait un appel à une procédure, cette procédure est appelée automatiquement quand le dialogue est lancé, avant même qu'il ne soit affiché. Cette façon de procéder est utilisée typiquement pour une mise à jour des composantes du dialogue avec des Alter Button …

```
declare sub diaghandlr
dialog Title "Dimensions fixées" width 100 height 50 calling diaghandlr
control OKbutton
sub diaghandlr
      note "Du dialogue"
end sub
```

D'abord ceci



puis le dialogue

Dialog_Handler particularités :

Une telle procédure ne peut pas faire appel à des données qui seraient recueillies par le dialogue puisque celui-ci n'est pas encore accessible. Comme de plus un handler (en fait, le nom d'une sous-routine) ne peut pas contenir d'arguments, si des données externes au handler sont nécessaires, il faut qu'elles soient communes au niveau de l'application; les variables impliquées doivent être DIMensionnées en dehors de toute sous-routine déclarée. Il n'est pas nécessaire qu'elles soient GLOBAL car nous n'avons pas à faire à des données à échanger entre deux applications via un DDE. Hormis cette circonstance particulière et le cas où l'application est faite de plusieurs modules MBO assemblés dans un projet MBP et partageant certaines variables, je déconseille totalement l'usage de GLOBAL pour déclarer une variable « commune » à plusieurs procédures (sous routines).

n.b. Ces particularités s'appliquent aussi à tous les « Control_Handlers », procédures appelées par un contrôle.

Les « Contrôles » : mots-clés

Les tableaux suivants regroupent toutes les conditions d'utilisation des mots-clés rencontrés dans les divers contrôles; les « valeurs » sont des mots en *italiques*.

Remarquons d'abord que tous les types de contrôle acceptent le mot-clé ID *ID*

Title *phrase*	
Statictext	oui
Edittext	oui
Listbox	oui, liste d'items séparés par "point-virgule"
Multilistbox	oui, liste d'items séparés par "point-virgule"
Popupmenu	oui, liste d'items séparés par "point-virgule"
Checkbox	oui
Radiogroup	oui, liste d'items séparés par "point-virgule"
Buttons (all)	oui
Style picker (tous)	non
DocumentWindow	non
GroupBox	oui

Title From Variable *nom_var*	
Statictext	non
Edittext	non
Listbox	Variable caractères dimensionnée au nombre d'items
Multilistbox	Variable caractères dimensionnée au nombre d'items
Popupmenu	Variable caractères dimensionnée au nombre d'items
Checkbox	non
Radiogroup	Variable caractères dimensionnée au nombre d'items
Buttons (all)	non
Style picker (tous)	non
DocumentWindow	non
GroupBox	non

Value *valeur*	
Statictext	non
Edittext	Une phrase initiale. Autrement le contrôle est vide.
Listbox	Nombre indiquant quel item est affiché sélectionné (1 pour le premier; 0 ou aucune valeur pour aucune sélection)
Multilistbox	Nombre indiquant quel item est affiché sélectionné; un seul item peut ainsi être initialement sélectionné (1 pour le premier; 0 ou aucune valeur pour aucune sélection)
Popupmenu	Nombre indiquant quel item est affiché sélectionné (1 pour le premier; 0 ou aucune valeur pour aucune sélection)
Checkbox	1 pour coché, 0 pour pas coché
Radiogroup	Nombre indiquant quel item est affiché sélectionné (1 pour le premier; 0 ou aucune valeur pour aucune sélection)
Buttons (all)	non
Style picker (tous)	Peut être une expression (Makexxx(…) … ou *var_xxx* variable du style correspondant au picker en question. Pen>MakePen, Symbol>MakeSymbol MakeFontSymbol MakeCustomSymbol, Brush>MakeBrush, Font>MakeFont

DocumentWindow	non
GroupBox	non

Into *variable*	
Statictext	non
Edittext	Variable caractères
Listbox	Variable SmallInt, indice de l'item sélectionné dans la liste
Multilistbox	Les différentes valeurs sélectionnées doivent être récupérées par une procédure spéciale. Voir >>>
Popupmenu	Variable SmallInt, indice de l'item sélectionné dans la liste
Checkbox	Variable logique (Vrai=1 pour coché, Faux=0 pour pas coché)
Radiogroup	Variable SmallInt, indice de l'item sélectionné dans la liste
Buttons (all)	non
Style picker (tous)	Variable de style (Pen, Symbol, Brush ou Font)
DocumentWindow	non
GroupBox	non

	Calling *control_handler*	Disable	Hide
Statictext	non	non	oui
Edittext	non	oui	oui
Listbox	oui	oui	oui
Multilistbox	oui	oui	oui
Popupmenu	oui	oui	non(*)
Checkbox	oui	oui	oui
Radiogroup	oui	oui	oui
Buttons (all)	oui	oui	oui
Style picker (tous)	oui	oui	oui
DocumentWindow	non	oui	oui
GroupBox	non	non	oui

(*) C'est ce que dit l'Aide de MapBasic, au moins jusqu'à la version 7. Je pense que c'est un oubli dans la description du contrôle parce que HIDE semble marcher correctement

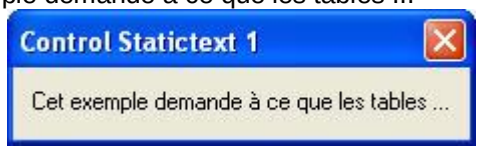
	Position (*x,y*)	Width *w*	Height *h*
Statictext	oui	oui	oui
Edittext	oui	oui	oui
Listbox	oui	oui	oui
Multilistbox	oui	oui	oui
Popupmenu	oui	oui	non
Checkbox	oui	non	oui
Radiogroup	oui	non	non
Buttons (all)	oui	oui	oui
Style picker (tous)	oui	oui	oui
DocumentWindow	oui	oui	oui
GroupBox	oui	oui	oui

Control Statictext

Statictext permet d'afficher une chaîne de caractères purement statique (aucune modification permise, aucune action associée)

Une seule ligne (non dimensionnée)

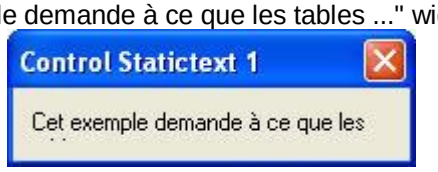
```
dialogTitle "ContrôleStatictext 1"
control statictext title "Cet exemple demande à ce que les tables ..."
```



La largeur du dialogue et celle de la boîte de texte sont définies implicitement par la longueur du « Title » du Statictext

Une seule ligne (dimensionnée en largeur)

```
Dialog Title "ContrôleStatictext 1"
Control statictext title "Cet exemple demande à ce que les tables ..." width 120
```



La largeur du texte (et celle du dialogue) est imposée; la ligne est tronquée de mots entiers

Une seule ligne (dimensionnée en hauteur)

```
Dialog Title "ContrôleStatictext 1"
Control statictext title "Cet exemple demande à ce que les tables ..." height 8
```

La hauteur minimale est de 8

La hauteur non spécifiée est 10

Multiligne (non dimensionnée)

```
Dialog Title "ContrôleStatictext 1"
Control statictext title "Cet exemple demande à ce que les tables ..."
      +chr$(13)+"fournies avc MapInfo doivent être accessibles"
```

La largeur est calculée pour recevoir tout le texte, sans tenir compte du chr$(13) qui force une nouvelle ligne; la seconde ligne est affichée en dessous mais est presque entièrement cachée

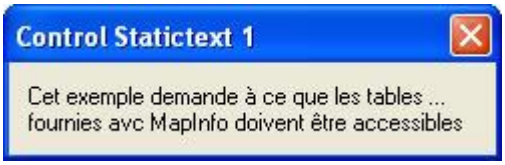
Multiligne (dimensionnée en hauteur)

```
Dialog Title "ContrôleStatictext 1"
Control statictext title "Cet exemple demande à ce que les tables ..."
    +chr$(13)+"fournies avec MapInfo doivent être accessibles" height 16
```

La hauteur est de 16 UD, valeur minimale pour afficher 2 lignes. La largeur n'est toujours pas bien calculée. Par contre, supprimer le chr$(13) donnerait le résultat assez semblable à celui qui suit mais sans le même ajustement dans la fin de ligne.

Multiligne (dimensionnée en hauteur et largeur)

```
Dialog Title "ContrôleStatictext 1"
Control statictext title "Cet exemple demande à ce que les tables ..."
    +chr$(13)+"fournies avec MapInfo doivent être accessibles" height 16 width 150
```



Pour un texte multiligne, il faut donc spécifier largeur ET hauteur.
Le retour de ligne n'est nécessaire que pour imposer un saut de ligne

Dimensionnement initial avant un Alter Control

Les éléments d'un dialogue sont souvent modifiés en recevant d'autres valeurs en cours d'exécution, mais ce qui est permis dans un tel « Alter Control » ne couvre pas les dimensions d'un StaticText (voir « Alter Control » p.50). Il faut donc prévoir lors de la création d'un dialogue l'espace suffisant pour pouvoir par la suite y afficher les autres textes prévus.

Nous savons qu'il y a deux techniques de dimensionnement, l'une implicite laisse MI décider de la longueur d'une boîte de texte par la phrase à y placer et ne marche que pour une simple ligne; l'autre, explicite, spécifie les dimensions, largeur et hauteur, et permet de définir des boîtes multilignes.

Une définition implicite d'un statictext à ligne simple doit donc contenir un « title " " » avec suffisamment d'espaces entre les " " pour recevoir la phrase maximale, tout en se rappelant que les caractères de la police utilisée dans les dialogues est de taille variable et que les espaces sont plus petits que la moyenne des lettres. C'est donc une solution relativement hasardeuse.

Control Edittext

Le contenu de cette boîte n'est limité que par la taille de la variable « string » qui la recevrait implicitement (32767 octets). La largeur n'établit donc que la portion visible du texte qui se déroule horizontalement. Si la hauteur est fixée à 20 UD ou plus, deux ou plusieurs lignes de texte seront visibles et se dérouleront verticalement au fur et à mesure du remplissage de la largeur de la boîte par du nouveau texte.

```
Dialog Title "ContrôleStatictext 1"
Control statictext title "Cet exemple demande à ce que les tables ..."
      height 8 position 5,5
control edittext position 150,3
control edittext position 150,15 width 80
```

Géométrie

Hauteur par défaut 12, minimale 12 pour une ligne
Mutliligne 12 + 8*nombre de lignes en plus de 1

Largeur par défaut 80, minimale 4 (un caractère) ?

Positionnement relatif à un Statictext pour un bon alignement des textes:
Y(edittext) = Y(statictext) – 2 UD

Distance verticale minimale entre 2 Statictext 12 UD

Valeur initiale et nouvelle valeur

Le mot clé VALUE définit la valeur affichée à l'ouverture du dialogue; la nouvelle valeur peut être assignée par le mot clé INTO à une variable.

La valeur initiale fixe doit être sous la forme d'une chaîne de caractères entre guillemets "chaîne initiale" ou d'un nombre (sans guillemets) utilisant exclusivement le point comme séparateur décimal éventuel. La valeur initiale peut aussi être une variable alphabétique ou numérique. Une variable numérique est affichée en respectant les définitions de l'installation Windows (séparateur de milliers, décimal …).

Maintien des valeurs saisies

Il est souvent souhaitable de présenter à l'utilisateur comme valeur initiale la dernière valeur qu'il a choisie; ceci peut simplifier son travail surtout quand le dialogue est suivi d'une phase de vérification qui peut réafficher le dialogue en cas de corrections

nécessaires. La valeur initiale est alors sous la forme d'une variable qui est la même que celle définie suivant INTO. Avant le début d'une telle boucle la variable *alpha_var* est initialisée « vide » puis EditText contient « VALUE *alpha_var* INTO *alpha_var* ». (voir aussi « Dialog Preserve » p. 52)

```
Dim alpha_var as string
alpha_var= ""
boucle:
Dialog Title "test101"
Control statictext title "Entrer une phrase d'au plus 25 caractères" position 5,5
Control edittext value alpha_var into alpha_var position 5,15
Control okbutton
If not commandinfo(1) then exit sub end if
If len(alpha_var) >25 then
    Note "Phrase trop longue; raccourcissez-la de "+str$(len(alpha_var)-25)
    Goto boucle
End if
Note "Phrase acceptée >"+alpha_var+"<"
```

Saisie de données numériques

L'utilisation du mot-clé INTO permet d'assigner une valeur numérique directement à une variable décimale (Integer, Smallint ou Float). MI est assez bon à ce travail mais il a certaines limitations.

Représentation des nombres

Tout d'abord en ce qui concerne la valeur initiale assignée au contrôle par VALUE, le nombre décimal doit être de la forme « 12345.67 » c'est-à-dire fait uniquement de chiffre et d'un point (éventuel). Pas d'espace, pas de virgule. Ce qui est dérangeant est la double représentation que MI fait de ce nombre décimal. Dans le dialogue, ce nombre peut être affiché comme « 12 345,67 », respectant en cela les paramètres régionaux de Windows, mais dans une NOTE il conserve la forme initiale. Toute entrée dans EditText doit être faite en respectant la forme compacte (pas d'espaces) et peut contenir un point ou une virgule comme séparateur décimal, une variante de plus dans les règles

En ce qui concerne un nombre entier, la même dualité de représentation existe et les contraintes se limitent à l'absence d'espace de milliers dans le nombre à entrer.

Conversion automatique

MI fait la conversion de la chaîne de caractères entrée comme un nombre en valeur numérique en ayant recours à un équivalent en langage plus performant de la fonction VAL(). Une telle fonction transforme la chaîne caractère par caractère en commençant par le premier à gauche (qui peut être – ou +) en chiffre assemblé progressivement en un nombre jusqu'à ce qu'il rencontre un caractère indésirable, autre qu'un chiffre et le point ou la virgule (le point seulement avec VAL()). Le reste de la chaîne est ignoré à compter du premier caractère indésirable. De plus si la chaîne commence par un de ces caractères, MI retourne une erreur.

Il est donc évident que d'avoir recours à une telle conversion automatique peut être dangereux, une erreur étant fatale pour l'application et une conversion incomplète n'étant pas signalée. Ceci est une bonne raison d'éviter autant que possible cette façon de procéder.

Une trappe systématique

Au lieu de définir la variable associée avec INTO comme décimale, elle est définie comme une chaîne; comme un control_handler ne peut pas être assigné à un EditText, nous devons créer un bouton qui convertira la chaîne avec VAL() et mettra à jour le contrôle avec la valeur convertie qui sera stockée dans une variable « float » commune. Si plusieurs valeurs sont entrées, toutes les vérifications peuvent se faire d'un coup et il est bon de ne sortir du dialogue qu'après vérification d'où une limitation d'accès possible à imposer au OKButton. L'utilisateur peut alors bien voir ce que MI a compris de ce qu'il vient d'entrer.

Cependant, la sécurité est relative en ce qui concerne l'accès au OKButton car il est opérationnel avant même que l'utilisateur fasse une correction. Il faudrait pour une sécurité optimale rajouter un autre bouton « Accepter » qui libérera OKButton et gèlera les entrées.

> *À noter que l'utilisation de la fonction VAL() entraîne que seul le point peut être utilisé comme séparateur décimal. Une note en ce sens aux utilisateurs serait probablement des plus utiles.*

Le petit programme suivant est un exemple simplifié de mise en œuvre.

```
declare sub main
declare sub convert
declare sub accept

dim f_val as float
dim a_val as string

sub main
dialoguetitle "test 2227"
contrôleedittext width 60 value "" into a_val  ID 1000
contrôlebutton title "Valider" calling convert
contrôlebutton title "Accepter" calling accept ID 1002 disable
contrôleokbutton ID 1001 disable
if not commandinfo(1) then exit sub end if
Note f_val
end sub

sub convert
f_val=val(readcontrolvalue(1000))
alter contrôle1000 value str$(f_val)
alter contrôle1002 enable
end sub

sub accept
alter contrôle1000 disable
alter contrôle1001 enable
end sub
```

Control ListBox

Listbox permet de choisir un item d'une liste. Nous allons traiter ici de géométrie, de conception de la liste et de sa construction, et de récupération et utilisation du choix.

Géométrie

Si la liste contient plus d'items que le nombre de lignes qui peuvent être affichées dans la boîte, elle est transformée en liste déroulante avec une barre/curseur prise dans la largeur de la boîte sur sa droite.

Liste de 4 items
Hauteur de 36

Liste de plus
De 4 items

Width *W*

Par défault, la largeur *W* est de 80 UD

Height *H*

Par défaut, la hauteur est de 70 UD et permet d'afficher 8 lignes. La hauteur pour afficher exactement n lignes est de (8*n + 4). 68 serait donc une hauteur plus juste et en effet avec cette valeur, le haut de la neuvième ligne n'apparaît plus.

Conception et construction de la liste

La liste est spécifiée comme étant le contenu de TITLE. Il y a deux façons de l'inclure :

A - comme une chaîne de caractères, les items étant séparés par des points virgules. Cette chaîne peut être écrite dans le code du contrôle (entre guillemets) ou par le biais d'une variable caractères.

```
... TITLE "un;deux;trois"
```

ou bien

```
Dim a as string
a="un;deux;trois"
... TITLE a
```

B – comme une variable caractères dimensionnée, chaque élément représentant un item de la liste ayant donc la forme d'un vecteur. Le mot clé devient alors plus complexe, TITLE FROM VARIABLE.

```
Dim a(3) as string
a(1)= "un "
a(2)="deux"
a(3)= "trois"
… TITLE FROM VARIABLE a
```

Si la liste contient des valeurs connues à l'avance et fixes, chaîne ou vecteur peut se coder directement. Par contre, si le contenu est variable tant au point de vue du nombre que des épelés, il faut construire cette chaîne ou ce vecteur.

Dans l'exemple qui suit, nous allons simplement préparer une liste des tables ouvertes parmi lesquelles l'utilisateur devra choisir. Nous allons construire les deux solutions successivement mais il serait bien simple de le combiner en une seule opération. La solution variable simple d'abord.

```
dim n_tabs, i as smallint
dim a_list as string
n_tabs=numtables()
if n_tabs=0 then exit sub end if
for i=1 to n_tabs
 a_list=a_list+tableinfo(i, 1)=";"
next
a_list=left$(a_list, len(a_list)-1)
```

La quatrième ligne (if …) est une trappe évitant tout risque. La dernière ligne retire le dernier point virgule; ce n'est pas indispensable mais s'il y en a un, la liste pourrait comprendre un item de plus si le point virgule était suivi d'un blanc; cet item vide pourrait alors être choisi et causer des problèmes.

La solution variable vecteur n'est pas plus compliquée

```
dim n_tabs, i as smallint
dim a_vect() as string
n_tabs=numtables()
if n_tabs=0 then exit sub end if
redim a_vect(n_tabs)
for i=1 to n_tabs
 a_vect(i)=tableinfo(i, 1)=";"
next
```

Le choix entre les deux solutions dépend en grande partie de la façon dont la sélection faite par l'utilisateur est récupérée et va être utilisée.

Récupération et utilisation du choix

Le choix se fait d'abord par une offre présentée à l'utilisateur par VALUE *i_value* (valeur ou variable smallint) et ensuite par le choix fait par lui par INTO *i_choix* (variable

smallint). Ces valeurs sont des positions dans la liste des items. Si i_value=0 ou VALUE n'est pas présent, la liste des items est affichée en commençant par le premier et aucun item n'est présélectionné. Une valeur positive indique un choix prédéterminé; la liste est affichée de façon à ce que ce choix soit visible.

VALUE 0, ou
mot clé absent

VALUE 5

La récupération du choix se fait donc par le biais de la variable *i_choix*, dont la valeur est disponible lorsque le dialogue est fermé ou pendant qu'il est encore ouvert avec un control_handler contenant un ReadControlValue(*ID*). *i_choix* a la valeur de 0 si aucun choix n'est fait. Cecei peut permettre de construire une trappe pour exiger un choix de l'utilisateur ou son abandon complet.

L'utilisation qui est faite de ce choix dépend naturellement de la suite des opérations. Examinons plusieurs situations :

1 – la liste des items est fixe et connue d'avance; le choix en tant qu'indice peut être utilisé dans le traitement qui suit. Les items sont alors présentés comme une chaîne simple. On peut alors utiliser la variable indice directement, comme par exemple

```
do case i_choix
        case 1
                ....
        case 2
                ...
        ...
end case
```

2 - la liste des items est fixe et connue d'avance; c'est le libellé du choix qui va servir dans le traitement. Si les items sont présentés comme une chaîne simple, il faut retrouver le libellé correspondant à la position choisie, ce qui nécessite l'utilisation d'une fonction spéciale. Si les items sont contenus dans une variable vecteur *i_vect()*, le libellé est simplement la valeur de *i_vect(i_choix)*.

La routine suivante retourne dans *item* le libellé du *ipos*^{éme} item dans la liste *sStr* utilisant le délimiteur *sToken*. Le séparateur peut être n'importe quel caractère ou chaîne de caractères puisqu'il est passé en argument.

```
sub StringToItem (
        byval sStr as string,       'Items en une chaîne délimitée
        item as string,             'item dans la position recherchée comme une variable String
        byval ipos as smallint,     'position de l'item comme smallint
        byval sToken as string)     'séparateur des items dans sStr
```

```
dim i, j, p0, p1 as smallint
item=""
j = len(sToken)
p0 = 1
p1 = instr(p0, sStr, sToken)
do while p1 > 0
        i = i + 1
        if i=ipos then
                item = mid$(sStr, p0, p1-p0)
                exit sub
        end if
        p0 = p1 + j
        p1 = instr (p0, sStr, sToken)
loop
if p0 < len(sStr) then
        i = i + 1
        if i=ipos then
                item = mid$(sStr, p0, Len(sStr)-p0+1)
                exit sub
        end if
end if
end sub
```

Si la valeur *ipos* de position n'est pas trouvée dans la liste, la chaîne *item* reste vide. Celle-ci peut être donc être utilisée pour déclencher une trappe d'alerte.

La plus grande complexité de code nécessaire pour convertir chaîne en libellé d'item prêche en faveur de la solution vectorielle d'entrée.

3 – la liste des items est variable et doit être reconstruite chaque fois (voir plus haut une façon de la construire); le choix entre chaîne et vecteur se présente sur la même base quoique certains modes de traitement ne sont plus imaginables; il est difficile de concevoir un « Do Case » avec un nombre variable de branches, mais par contre, l'*indice* de choix d'une table ouverte peut être utilisé directement dans un énoncé comme TableInfo(*indice*,…)

Utilisation du « double-clic » et de « TriggerControl() »

Si un dialogue ne contient qu'une liste, l'utilisateur doit d'abord choisir un item par un clic, puis cliquer sur OK pour fermer le dialogue. La technique du double-clic sur l'item voulu permet de fermer aussi le dialogue. La liste doit avoir son control_handler qui détecte le double-clic, enregistre le choix et ferme le dialogue.

Dans le code qui suit la fermeture programmée du dialogue ne serait pas reconnue par un « if commandinfo(1) » après le dialogue comme équivalent à la fermeture par un OKButton. Il nous a fallu rajouter une variable logique en commun qui est mise à jour par le list_handler pour détecter que le double-clic a fermé le dialogue dans des conditions acceptables.

```
declare sub list_handler
dim i_choix as smallint
dim double as logical
```

```
dialog Title "Double clic"
    control listbox title "un;deux;trois;quatre" into i_choix ID 1000 calling list_handler
    control okbutton
if not commandinfo(1) then
    if not double then
        exit sub
    end if
end if
note"Choix "+str$(i_choix)
sub list_handler
double=0
    if commandinfo(1) then
        i_choix=readcontrolvalue(1000)
        dialog remove
        double=1
    end if
end sub
```

S'il y a plusieurs listes dans le même dialogue, chaque liste peut avoir son propre handler pour ce qui est du double-clic ou partager le même. Si le handler est commun à plusieurs contrôles, de façon à détecter dans quelle liste le double-clic a eu lieu, il faut d'abord utiliser la fonction « TriggerControl() » qui retourne l'ID du dernier contrôle actionné par l'utilisateur.

Comme il faut savoir de quelle liste provient le choix, on maintient l'ID du contrôle choisi en commun. Cet exemple est un peu farfelu, car il ne gère pas les choix exprimés par simple clic et OK; il pourrait y avoir un choix dans les deux listes, seul celui du deuxième contrôle est enregistré mais pas l'ID du contrôle. Le but de ce code est essentiellement de montrer l'usage de la fonction TriggerControl()

```
declare sub list_handler
dim i_choix, i_list as smallint
dim double as logical
dialog Title "Double clic"
    control listbox title "un;deux;trois;quatre" into i_choix
        ID 1000 calling list_handler position 10,10 height 36
    control listbox title "un et demi;deux et demi;trois et demi;quatre et demi" into i_choix
        ID 1001 calling list_handler position 100,10 height 36
    control okbutton
if not commandinfo(1) then
    if not double then
        exit sub
    end if
end if
note"Choix "+str$(i_choix)+" dans "+str$(i_list)

sub list_handler
double=0
    if commandinfo(1) then
        i_list=triggercontrol()
        i_choix=readcontrolvalue(i_list)
        dialog remove
        double=1
    end if
end sub
```

Control MultiListBox

MultiListBox permet de choisir un ou plusieurs items d'une liste. La géométrie et la conception de la liste et sa construction sont identiques aux cas de la ListBox, mais la question de la récupération des choix est très différente.

Nous devons d'abord préciser qu'avec une MultiListBox on ne peut présélectionner qu'une seule valeur VALUE *i_value*

Récupération des choix

On ne peut récupérer les choix faits dans une MultiListBox que par le biais d'un control_handler qu'il faut ajouter à un autre contrôle, le plus souvent à OKButton, et en respectant une technique relativement simple. Il faut lire le contenu du contrôle en utilisant la fonction ReadControlValue() de façon répétitive. Le premier appel à cette fonction retourne le premier choix, le deuxième appel le deuxième choix, jusqu'à ce que l'indice de choix soit zéro.

```
declare sub choix_multi
dim l_list as string

l_list="un;deux;trois;quatre;cinq;six;sept;huit;neuf"
dialog title "multilistbox"
    control multilistbox title l_list  ID 1008
    control okbutton calling choix_multi

sub choix_multi
dim n_choix, i, m_choix() as smallint
dim l_choix as string
l_choix=""
boucle:
    n_choix=readcontrolvalue(1008)
    if n_choix=0 then goto fin end if
    l_choix=l_choix+str$(n_choix)+";"
    goto boucle
fin:
note left$(l_choix, len(l_choix)-1)
end sub
```

Dans cet exemple, on se contente de reconstruire une chaîne des indices des choix. On pourrait imaginer tout autre traitement possible.

Une autre possibilité souvent mentionnée est d'utiliser une boucle While … Wend. Dans ce cas particulier, comme la condition de la boucle est la valeur du contrôle, il faut prendre des précautions pour n'appeler les Readcontrolvalue() qu'à bon escient sinon on manquerait des valeurs. À mes yeux, le code suivant est bien plus complexe que la simple boucle précédente, mais à vous de juger.

```
declare sub choix_multi
dim l_list as string

l_list="un;deux;trois;quatre;cinq;six;sept;huit;neuf"
dialog title "multilistbox"
	control multilistbox title l_list ID 1008
	control okbutton calling choix_multi

sub choix_multi
dim l_choix as string
dim i_choix as smallint
l_choix=""
i_choix=1
while i_choix>0
	i_choix=readcontrolvalue(1008)
	if i_choix=0 then goto fin end if
	l_choix=l_choix+str$(i_choix)+";"
wend
fin:
note left$(l_choix, len(l_choix)-1)
end sub
```

Control PopUpMenu

L'utilisateur exerce son choix à partir d'une liste présentée seulement comme un seul item mais qui s'ouvre quand on clique sur le contrôle. La liste peut afficher jusqu'à 10 items avant de devenir déroulante. L'item présélectionné par VALUE est affiché dans la boîte permanente. La similarité de fonctionnement avec ListBox est complète.

La seule différence est dans la définition géométrique : le mot clé HEIGHT même s'il est reconnu n'est pas nécessaire car il n'affecte en rien la taille du dialogue; cependant, s'il est utilisé, une réserve verticale correspondante est créée (voir de combien le contrôle suivant qui serait positionné automatiquement est éloigné du PopUpMenu).

Le mot clé WIDTH semble devoir être défini car la largeur utile de la boîte est diminuée de la flèche d'activation de la liste et la tirette de déplacement si elle existe est comptée dans la largeur de la liste. (La valeur par défaut de 40 a été utilisée dans les illustrations)

```
dim l_list as string
dim i_choix as smallint
l_list="un;deux;trois;quatre;cinq;six;sept;huit;neuf;dix;onze;douze;treize"
dialog title "popupmenu"
    control popupmenu title from variable l_list value 3 into i_choix
    control okbutton
```

L'item 3 est présélectionné

En cliquant sur la petite flèche de droite, la liste s'ouvre (10 items)

Control CheckBox

Une simple case que l'utilisateur coche ou décoche pour exprimer son choix.

La géométrie est simple; hormis la position, seul le mot clé WIDTH a un impact. Tout comme avec EditText, il peut être omis pour laisser MI décider de la largeur mais on ne peut pas afficher le titre sur plusieurs lignes, HEIGHT n'étant pas reconnu à cette fin.

Si VALUE n'est pas utilisé, la case est cochée. La valeur initiale est 1 (coché) ou 0 (pas coché) ou une variable logique T (vrai = 1) F (faux =0). INTO est utilisé avec une variable logique.

```
dim i_choix, j_choix as logical
dialog title "checkbox"
    control checkbox title "Afficher Légende" into i_choix
    control checkbox title "Calculer moyennes" value 0 into j_choix
    control okbutton
```

Dialogue à son ouverture

RadioGroup

Ce contrôle permet de choisir une possibilité parmi celles offertes en sélectionnant une des puces qu'il crée. Comme les spécifications de TITLE, VALUE et INTO sont les mêmes que pour ListBox, ce contrôle a donc la même fonctionnalité. Leur différence tient essentiellement dans l'impact visuel; tous les choix d'un RadioGroup sont visibles d'un coup d'œil, même pour les listes dépassant les 10 items.

La hauteur est définie automatiquement par MapInfo, en fonction du nombre d'items. Si l'Aide de MapBasic ne mentionne pas WIDTH, ce mot clé est pourtant opérationnel. Comme on risque de tronquer le libellé d'un item si la largeur donnée n'est pas assez grande, il faut avoir une bonne raison de l'utiliser; ce pourrait être le cas où l'on dispose d'un espace restreint dans un dialogue et que l'on veuille éviter une interférence avec un dialogue voisin, surtout si la liste est composée d'items dont la longueur n'est pas forcément connue d'avance.

```
dim l_list as string
dim i_choix as smallint
l_list="un beaucoup plus long que les autres;deux;trois;quatre" '
dialog title "radiogroup"
      control radiogroup title from variable l_list value 3 into i_choix width 60
```

Choix initial 3

Control Buttons :
OKButton, CancelButton, Button

Les boutons permettent de lancer des opérations qui leur sont attachées implicitement dans leur nature ou explicitement grâce à un control_handler.

OKButton et CancelButton ont la fonction implicite de fermer le dialogue mais ils procèdent différemment. OKButton met à jour toutes les variables définies par des INTO avec le contenu du contrôle correspondant alors que CancelButton ne le fait pas. Un Button, ou bouton standard, n'a aucune fonction implicite

On peut attacher un control_handler aux 3 types de boutons

Géométrie

OKButton et CancelButton viennent avec une taille prédéterminée, mais qui peut être changée si besoin est. Les dimensions par défaut semblent être pour les deux WIDTH 38 HEIGHT 14. La hauteur pourrait être réduite à 12; moins que cela et le cadre interne de « contrôle actif » couvre en partie le texte. Cette remarque sur la hauteur s'applique aussi aux boutons standards.

La largeur d'un bouton standard doit tenir compte de la longueur du texte de son TITLE. Si WIDTH n'est pas spécifiée, MI la calcule avec une certaine marge, donc en utilisant WIDTH, on peut réduire la largeur automatique de 6 UD. Si la largeur du bouton est trop petite, le texte sera affiché tronqué aux deux bouts; à la différence d'un StaticText dans lequel le texte est aligné à gauche, dans un bouton, il est centré.

```
Dialog Title "boutons"
Control button position 10,0 title "Bouton standard"
Control button position 10,20 title "Bouton standard" width 40 height 10
Control okbutton
```

Bouton du haut dimensionné automatiquement.
Bouton du bas trop petit dans les 2 directions
Le cadre de « contrôle actif » brouille le texte

Control Style Pickers : BrushPicker, FontPicker, PenPicker et SymbolPicker

La différence essentielle entre ces contrôles est le style auquel ils font appel

Ils supportent tous les mots clés de positionnement et de dimensionnement, HIDE et DISABLE, VALUE et INTO. Ils refusent tous TITLE.

Valeur initiale et nouvelle valeur

Il doit y avoir une correspondance parfaite entre le type de contrôle et le style de ces valeurs.

La valeur initiale suivant VALUE est une expression de style, c'est-à-dire une formule de création de style (MakeBrush, …) ou une variable de style.

La nouvelle valeur doit être récupérée dans une variable de style suivant INTO

Géométrie

Il ne semble pas y avoir de contraintes particulières concernant les dimensions minimales et maximales. En dessous de 10, l'image n'est plus trop lisible. Les dimensions par défaut donnent un carré de 20x20 UD. Un contrôle qui n'est pas carré va entraîner une certaine déformation de l'image affichée seulement pour PenPicker.

L'absence de contraintes nous permettra d'afficher des « symboles » qui pourraient être considérés comme des logos.

Affichage de logos

Jusqu'à l'apparition de la version 7, la seule façon connue d'afficher un logo était d'utiliser un SymbolPicker pour afficher un logo dans un dialogue. La préparation est bien précise. Comme il y 3 possibilités de créer un symbole, il pourrait y avoir trois préparations différentes, mais il n'y en a que deux car avec MakeSymbol on est limité aux seuls symboles de MapInfo 3.

> MakeFontSymbol : il faut avoir une police avec le symbole voulu, sinon il faut en créer une ou rajouter un glyphe dans une existante déjà. Il faut bien s'assurer que la police est capable d'un affichage graphique.

MakeCustomSymbol : le logo doit être sous la forme voulue pour être reconnu par MapInfo (nom de fichier au maximum 31 caractères, format BMP Windows, image carrée, 16 couleurs, taille maximale du fichier 128 ko au moins jusqu'à la version 7.0) et placé dans le répertoire CustSymb enterré quelque part.
(utilisez LocateFile$(8) pour en trouver la place exacte). Ne pas oublier de lancer un « Reload Symbols » après un ajout dans ce répertoire pour que MI reconnaisse l'addition s'il est ouvert; il la reconnaîtrait de toute façon mais lors de son prochain lancement seulement.

L'affichage est défini de la façon suivante :

A – positionner et dimensionner le contrôle

B – VALUE = Make… pour le symbole voulu; sa taille devra être ajustée pour remplir au mieux le contrôle dont les dimensions peuvent aussi être ajustées dans ce même sens.

C- Ajouter le mot clé DISABLE pour être sûr que le contenu du contrôle ne sera pas modifié.

Dans tous les cas, la taille d'un Symbol est limitée à 48 points au moment de l'affichage. Remarquer que pour avoir de la couleur, il faut utiliser la fonction RGB () dans le MakeFontSymbol, autrement le symbole reste noir.

```
dialog Title "Style Pickers" width 74
control symbolpicker position 10,10 height 50 width 54
        value makefontsymbol(94,RGB(255,0,0),48,"Wingdings",17,0) disable
control symbolpicker position 10,60 width 54 height 45
        value makecustomsymbol("GLOB1-32.BMP",1,48,0)disable
control okbutton position 19,116
```

FontSymbol en haut
CustomSymbol en bas

C'est certainement une façon de mettre de la couleur dans un dialogue. Certains se plaignent de la présence d'un cadre impossible à effacer avec cette technique. Plus sur le sujet à la toute fin du document « BMP et custom symbols ».

Control GroupBox

Ce contrôle a une fonction purement esthétique puisque seuls les mots clés TITLE, POSITION, WIDTH, HEIGHT et HIDE sont reconnus.

Pour être affiché, le contrôle doit être défini par les valeurs des mots clés WIDTH et HEIGHT

Le TITLE est affiché en entier quelle que soit la largeur donnée.

Géométrie

La hauteur minimale est de 11 UD. Plus petite, le texte commence à cacher la ligne du bas.

La zone utilisable à l'intérieur d'un GroupBox est réduite en hauteur par la présence du texte. De plus dans les deux directions on peut vouloir décoller un peu les composantes du cadre de la boîte

```
Dialog Title "GroupBox"
Control GroupBox position 10,10 title "Groupe 1" width 42 height 11
Control GroupBox position 10,30 title "Groupe 2 très prolongé" width 60 height 50
Control GroupBox position 12,36 width 56 height 42
Control okbutton
```

Groupe 1 est une boîte minimale (Utilité ?)
Le 2éme titre est trop long pour sa boîte, même pour le dialogue
La boîte encastrée dans le « Groupe 2 »
délimite la zone utile pour placer des composantes

Control DocumentWindow

Version 7.0 et plus seulement

Le contrôle DocumentWindow permet d'afficher dans une boîte le contenu d'une fenêtre-document de MapInfo (carte, tableau, légende, graphique, mise en page). Par le biais de cette technique, des illustrations comme des logos ou des images peuvent être aussi incluses dans un dialogue du moment que les sources sont enregistrées comme une table MapInfo affichable dans une fenêtre carte.

Code requis

L'utilisation de ce contrôle requiert la création d'un dialog_handler qui va créer la fenêtre dont le contenu va apparaître dans la boîte. Dans le dialogue, nous allons avoir l'appel au dialog_handler et la définition standard du contrôle

```
Dialog Title "Logo intégré" calling dialoghandler
    Control DocumentWindow ID 1007 Width 110 Height 100 position 10,10
    Control okbutton
```

Dans le dialog_handler, les éléments requis sont

```
Dim iHwnd As Integer
iHwnd = ReadControlValue(1007)
Set Next Document Parent iHwnd Style 1
```

La suite est faite des énoncés définissant la fenêtre, par exemple pour une carte MapInfo :

```
open table "croixbask.tab"
map from croixbask
set coordsys table croixbask
set window frontwindow() width 1.6 Units "in" height 1.6 Units "in" scrollbars off
set map window frontwindow() center (0,0) zoom 21.5 Units "cm"
```

Ces lignes font l'hypothèse que la table MapInfo est placée dans le répertoire de l'application. Cette hypothèse est bien fondée car si l'application doit être distribuée, les fichiers nécessaires pour ce contrôle doivent l'être aussi, et garder tous les fichiers relatifs à la même application dans le même répertoire est bien sensé.

Remarquer que les trois énoncés « Set… » permettent d'ajuster précisément les dimensions de la carte fenêtre et son contenu. Ces paramètres doivent être établis en tenant compte de la forme de la fenêtre (il est préférable que les deux soient dans la même proportion) mais l'exemple de la photo met cette proposition en doute.

Paramétrisation

Comme l'existence d'une telle fenêtre-document dans un dialogue semble être indépendante de ce qui se passe dans la fenêtre MapInfo, on peut « préparer » la fenêtre-carte qui peut rester ouverte sans problème et utiliser des outils MI comme

« change view » pour obtenir les valeurs d'intérêt (zoom, centre de la carte affichée) ou un outil comme WinMastr[1] pour dimensionner exactement la fenêtre.

Un élément essentiel de ce contrôle est la place qu'il occupe. L'image affichée peut n'occuper qu'une partie de l'espace réservé calculé « secrètement » par MI. On peut s'en rendre compte en ne positionnant pas le contrôle OKButton. On peut le trouver nettement plus bas ou plus à droite que prévu ce qui dénote une réserve plus grande que l'image. Il faut alors tâtonner pour essayer de resserrer les limites comme dans l'exemple de la photo.

Sécurité

Une carte MI affichée dans un contrôle DocumentWindow est ouverte à toutes modifications possibles. Par le biais d'un clic droit de la souris dans le contrôle, on peut ouvrir le contrôleur de pile et rendre la couche éditable.

Si l'on veut maintenir l'intégrité de la carte, il faut prendre alors la précaution de rendre le contrôle inactif (mot clé Disable) ce qui entraîne aussi la perte d'interactivité de ce contrôle (consultation, déplacement du contenu …)

Précautions préliminaires

Comme l'ouverture d'une table qui n'existe pas (mauvais nom, mauvaise route) entraîne une erreur et l'arrêt de l'application, il est recommandé de rajouter du code pour intercepter cette erreur et exiger de placer la bonne table au bon endroit. Cette mesure peut sembler sévère mais si l'on juge que l'affichage de l'illustration est fondamental, il faut passer par là (Voir le code dans « Carte MapInfo comme logo » plus bas). Une solution moins stricte court-circuiterait seulement la construction de ce contrôle et donc son affichage (voir « Photo enregistrée comme .TAB »).

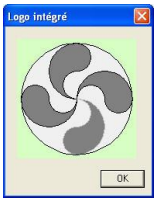
Après coup

Contrairement à ce que l'on pourrait penser après avoir « lu » que ce qui arrive dans la fenêtre-fille est indépendant de ce qui se passe dans MI (par exemple, la même carte peut être affichée dans MI et dans le dialogue), lors de la fermeture du dialogue la table utilisée comme base de l'illustration n'est pas fermée. Il faut donc prévoir qu'après la fermeture du dialogue, le code contienne un « Close Table … » pour ne pas laisser cette table ouverte.

[1] Outil MLC de l'auteur www.paris-pc-gis.com/decharge.htm

Quelques exemples de mise en œuvre

Carte MapInfo comme Logo



Solution préventive rigoureuse. Noter le « scrollbar off » totalement inutile avec un logo, et le fait que le contrôle soit désactivé pour éviter toute modification.

```
declare Sub DialogHandler

dialog Title "Logo intégré" calling dialoghandler
      Control DocumentWindow ID 1007 Width 110 Height 100 position 10,10
      Control okbutton

Sub DialogHandler
Dim iHwnd As Integer
if not fileexists(applicationdirectory$()+"croixbask.tab") then
      note "La Table MapInfo 'Croixbask ...' doit être dans le répertoire de cette application"+
            chr$(13)+chr$(13)+"Faites le changement avant de relancer l'application"
      end program
end if
iHwnd = ReadControlValue(1007)
Set Next Document Parent iHwnd Style 1
open table "croixbask.tab"
map from croixbask
set coordsys table croixbask
set window frontwindow() width 1.6 Units "in" height 1.6 Units "in" scrollbars off
set map window frontwindow() center (0,0) zoom 21.5 Units "cm"
Alter Control 1007 disable
End Sub
```

Photo (enregistrée com .tab)

Solution préventive douce.

Le problème réside dans la détermination des paramètres de taille. Ayant établi le centre de la photo et les dimensions pour ce que l'on veut voir, il faut trouver les dimensions pour la boîte. Celle-ci affiche le contenu de la fenêtre à compter du coin gauche supérieur. Il faut donc jouer sur la taille de la fenêtre si celle de la boîte est fixée.

De plus, les proportions largeur/hauteur pour la fenêtre et la boîte ne sont pas forcément les mêmes si on tient à afficher tout le contenu de la fenêtre. Les meilleurs résultats pour notre exemple ont été obtenus pour une boîte de 48x60 UD et une carte de .7/.9 pouce.

```
declare Sub DialogHandler

dialog Title "Image intégrée" calling dialoghandler
      Control DocumentWindow ID 1007 Width 48 Height 60 position 10,10
      Control okbutton 'position 10,98
```

```
Sub DialogHandler
Dim iHwnd As Integer
if not fileexists(applicationdirectory$()+"jakes_1.tab") then
      note "La Table MapInfo 'jakes_1 ...' devrait être dans le répertoire de cette application"+
            chr$(13)+chr$(13)+
            "Faites le changement pour voir cette illustration la prochaine fois."
      exit sub
end if
iHwnd = ReadControlValue(1007)
Set Next Document Parent iHwnd Style 1
open table "jakes_1.tab"
map from jakes_1
set coordsys table jakes_1
set window frontwindow() width .7 Units "in" height .9 Units "in" scrollbars off
set map window frontwindow() center (55,-70) zoom 100 Units "ft"
Alter Control 1007 disable
End Sub
```

Carte et tableau intégrés

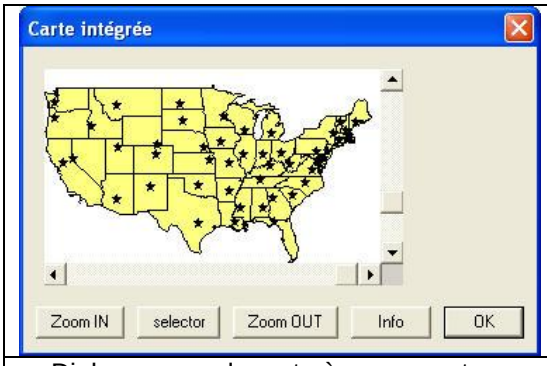
Aucune vérification sur la présence des tables.

La fenêtre MI est repositionnée et redimensionnée pour que les dialogues conservent leurs positions relatives prévues quelles que soient la position et la taille de la fenêtre MI.

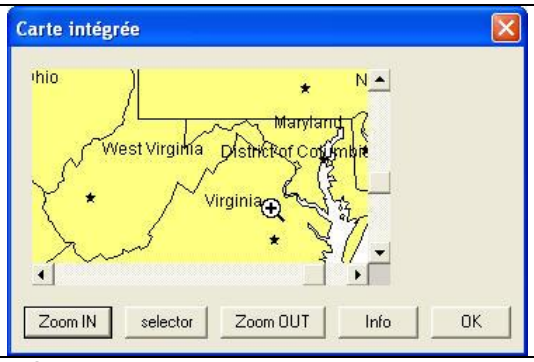
La fenêtre-carte est laissée accessible. Cela permet de faire des zooms et des déplacements. Comme les outils MapInfo ne sont pas accessibles directement (la présence du dialogue élimine toute action à l'extérieur de ses limites), j'ai rajouté 3 boutons pour faciliter les zooms (Voir comment simuler des icones juste après). Les étiquettes des états s'affichent à un certain zoom seulement (1000 km)

Cela permet aussi de faire une sélection. Si un état est sélectionné, le bouton « Info » affiche un tableau avec quelques variables pour le cas en question. Ce bouton ne réagit pas si une capitale est sélectionnée ou si aucune sélection n'est faite; des trappes pourraient être ajoutées pour avertir l'utilisateur. Le programme pourrait aussi détecter de quelle carte originale vient la sélection et choisir entre 2 dialogues d'affichage de tableau.

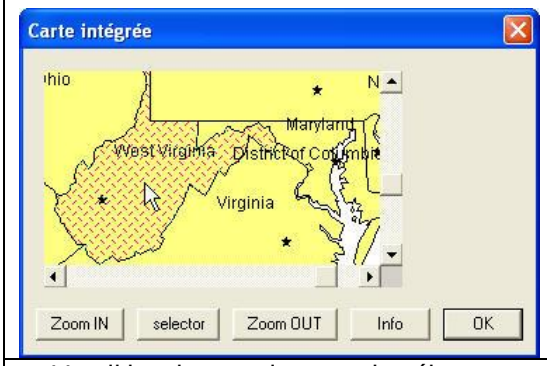
En fin de traitement, le OKButton du contrôle de la carte ferme les tables. Rien n'est prévu dans le cas d'un autre type de fermeture.



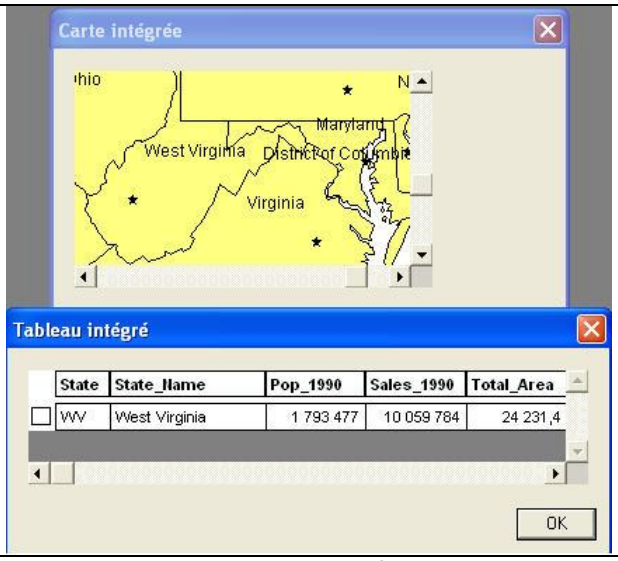
Dialogue avec la carte à son ouverture



Après un zoom (l'outil local est toujours visible)
les étiquettes apparaissent



L'outil local est maintenant le sélecteur
Un état est sélectionné



Cliquer sur info ouvre le deuxième dialogue avec
un tableau contenant quelques données

```
declare sub main
declare Sub DialogHandler
declare sub zoomin
declare sub selector
declare sub zoomout
declare sub infocas
declare Sub DialogHandler2
declare sub okout

sub main
set window 1011 position (2.80208, 2.42958) units "in" Width 10 units "in" Height 7.76042 units "in"
dialog Title "Carte intégrée" calling dialoghandler
      Control DocumentWindow ID 1007 Width 174 Height 100 position 10,10
      Control button title "Zoom IN" calling zoomin
      Control button title "selector" calling selector
      Control button title "Zoom OUT" calling zoomout
      Control button title "Info" calling infocas
      Control okbutton calling okout
end sub

Sub DialogHandler
```

```
Dim iHwnd As Integer
iHwnd = ReadControlValue(1007)
Set Next Document Parent iHwnd Style 1
open table "statecap.tab"
open table "states.tab"
map from statecap, states
set coordsys table states
set window frontwindow() width 2.66 Units "in" height 1.6 Units "in" scrollbars on
set map window frontwindow() layer 2 label position above center with State_Name auto on duplicates on
visibility zoom (0,1000) units "km"
set map redraw on
set map window frontwindow() center (-95,38) zoom 5200 Units "km"
End Sub

sub zoomin
run menu command 1705
end sub

sub selector
run menu command 1701
end sub

sub zoomout
run menu command 1706
end sub

sub infocas
dim nom as string
if selectioninfo(3)=0 then exit sub end if
if selectioninfo(1)<>"STATES" then exit sub end if
nom=selection.col1
select State, State_name, Pop_1990, Sales_1990, Total_Area from states where State=nom into sele
dialog Title "Tableau intégré" calling dialoghandler2 position 260, 390
      Control DocumentWindow ID 1007 Width 270 Height 56 position 10,10
      Control okbutton
end sub

Sub DialogHandler2
Dim iHwnd As Integer
iHwnd = ReadControlValue(1007)
Set Next Document Parent iHwnd Style 1
browse * from sele
set window frontwindow() width 4.16 Units "in" height .84 Units "in" scrollbars off
Alter Control 1007 disable
run menu command 304
End Sub

sub okout
close table states
close table statecap
end sub
```

Reconstruction d'une barre d'outils

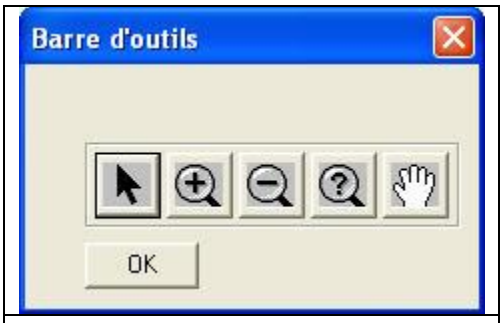
La question est parfois posée de la possibilité d'utiliser les icônes MapInfo dans un dialogue. La réponse traditionnelle est que ce n'est pas faisable. Cependant DocumentWindow nous permettrait de le faire à condition de les reconstruire.

Il faut d'abord avoir les images BMP des icônes voulues; dans l'exemple qui suit j'ai utilisé les grandes icônes (si on voulait utiliser le petit format, il faudrait recalibrer tous les paramètres). On peut toujours intégrer ces images dans un dialogue (voir plus haut). Cependant, pour rendre à ces images leur opérationnalité originale, il faudrait leur accrocher des control_handlers ce que ne permet pas DocumentWindow.

Une solution est d'encadrer les images dans des boutons qui eux vont supporter un control_handler. Pour rendre cette solution efficace, il faut neutraliser (disable) l'image même.

On peut même encadrer tous les boutons par un GroupBox et l'illusion est presque parfaite.

J'ai choisi de définir la position de l'ensemble par XX et YY, variables fixant la position du GroupBox. Cela permet de positionner la barre où l'on veut dans un dialogue, en prenant soin de détacher éventuellement le OKButton, sans avoir à calculer toutes les valeurs de positionnement.



La barre en position 20,20

```
declare sub dialoghandler
declare sub zoomplus
declare sub zoommoins
declare sub selector
declare sub changeview
declare sub grab

dim xx, yy as float
xx=20
yy=20
dialogueTitle "Barre d'outils" calling dialoghandler
      control groupbox position xx, yy width 5+24*5 height 30
      Control DocumentWindow ID 1006 Width 20 Height 18 position xx+6,yy+10 disable
      control button position xx+3,yy+7 width 22 height 20 title "" calling selector
      Control DocumentWindow ID 1007 Width 20 Height 18 position xx+6+24*1,yy+10 disable
      control button position xx+3+24*1,yy+7 width 22 height 20 title "" calling zoomplus
      Control DocumentWindow ID 1008 Width 20 Height 18 position xx+6+24*2,yy+10 disable
      control button position xx+3+24*2,yy+7 width 22 height 20 title "" calling zoommoins
```

```
    Control DocumentWindow ID 1009 Width 20 Height 18 position xx+6+24*3,yy+10 disable
    control button position xx+3+24*3,yy+7 width 22 height 20 title "" calling changeview
    Control DocumentWindow ID 1010 Width 20 Height 18 position xx+6+24*4,yy+10 disable
    control button position xx+3+24*4,yy+7 width 22 height 20 title "" calling grab
    control okbutton position xx, yy+35

sub dialoghandler
Dim iHwnd As Integer

iHwnd = ReadControlValue(1006)
Set Next Document Parent iHwnd Style 1
open table "gr01_000_26.tab"
map from gr01_000_26
set coordsys table gr01_000_26
set window frontwindow() width 1.2 Units "in" height 1.2 Units "in" scrollbars off
set map window frontwindow() center (60,-60) zoom 120 Units "ft"

iHwnd = ReadControlValue(1007)
Set Next Document Parent iHwnd Style 1
open table "gr01_004_26.tab"
map from gr01_004_26
set coordsys table gr01_004_26
set window frontwindow() width 1.2 Units "in" height 1.2 Units "in" scrollbars off
set map window frontwindow() center (60,-60) zoom 120 Units "ft"

iHwnd = ReadControlValue(1008)
Set Next Document Parent iHwnd Style 1
open table "gr01_005_26.tab"
map from gr01_005_26
set coordsys table gr01_005_26
set window frontwindow() width 1.2 Units "in" height 1.2 Units "in" scrollbars off
set map window frontwindow() center (60,-60) zoom 120 Units "ft"

iHwnd = ReadControlValue(1009)
Set Next Document Parent iHwnd Style 1
open table "gr01_006_26.tab"
map from gr01_006_26
set coordsys table gr01_006_26
set window frontwindow() width 1.2 Units "in" height 1.2 Units "in" scrollbars off
set map window frontwindow() center (60,-60) zoom 120 Units "ft"

iHwnd = ReadControlValue(1010)
Set Next Document Parent iHwnd Style 1
open table "gr01_007_26.tab"
map from gr01_007_26
set coordsys table gr01_007_26
set window frontwindow() width 1.2 Units "in" height 1.2 Units "in" scrollbars off
set map window frontwindow() center (60,-60) zoom 120 Units "ft"
end sub

sub selector
run menu command 1701
end sub

sub zoomplus
run menu command 1705
end sub

sub zoommoins
```

```
run menu command 1706
end sub

sub changeview
run menu command 805
end sub

sub grab
run menu command 1702
end sub
```

Procédure linéaire ou intégrée?

Quand on lance une application, il faut souvent commencer par saisir toute une variété de paramètres, nom de table, colonne, type de traitement etc. Il faut aussi souvent valider cette information dont le choix pour l'une peut conditionner les possibilités pour une suivante. Cette procédure d'acquisition peut prendre deux formes principales; la procédure linéaire procède par une séquence de questions (dialogues), chacune pouvant donner lieu à validation (avec messages possibles), certaines pouvant ouvrir un branchement particulier. Avec la procédure intégrée, un seul dialogue est offert à l'utilisateur qui doit répondre aux questions au fur et à mesure qu'elles deviennent accessibles.

Chaque procédure a ses avantages et inconvénients. La première est beaucoup plus facile à construire, ne requérant qu'un effort de design minimal et acceptant ajouts ou modifications sans trop de difficultés; l'information présentée à l'utilisateur dépend de ses choix précédents; il se contente de traiter une seule question ou réponse à la fois. La procédure intégrée est bien plus difficile à construire car il faut assembler dans un seul contenant des composantes diverses et créer un code qui permet d'y cheminer correctement; l'utilisateur a par contre une vue d'ensemble de ce qui l'attend et n'a pas à faire face à une multiplicité de fenêtres et de clics de souris pour atteindre la suivante.

Je n'ai pas de solution favorite sinon que je trouve plus facile dans la phase de développement d'un outil de travailler en linéaire, mais que le recours à un dialogue intégré peut se révéler un atout important dans la diffusion de l'outil. Pour vous permettre de juger, je vous propose un problème donné et les solutions que j'ai développées pour chaque type.

Énoncé de la commande :

Recueillir le nom d'une table ouverte et celui d'une colonne pour un type donné (Integer, Smallint ou String d'une largeur minimale donnée)

Solutions :

Vous trouverez d'abord les images des fenêtres impliquées pour chacune des procédures, puis le code pour les générer.

J'ai inclus dans le code toutes les définitions utiles pour que les fichiers MB soient compilables et exécutables.

Procédure linéaire

Vérification Préliminaire (procédure interrompue)

Étape 1 >>> Choix de la table

Interruptions de la procédure

Étape 2 >>> Définition du type de variable

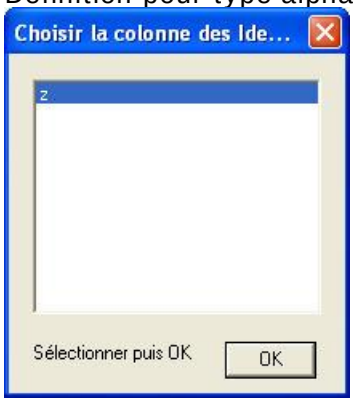
Aucune colonne Integer

Aucune colonne Smallint

Étape 2 bis Pour colonne alpha >>>

Définition pour type alpha

Aucune colonne alpha de largeur voulue

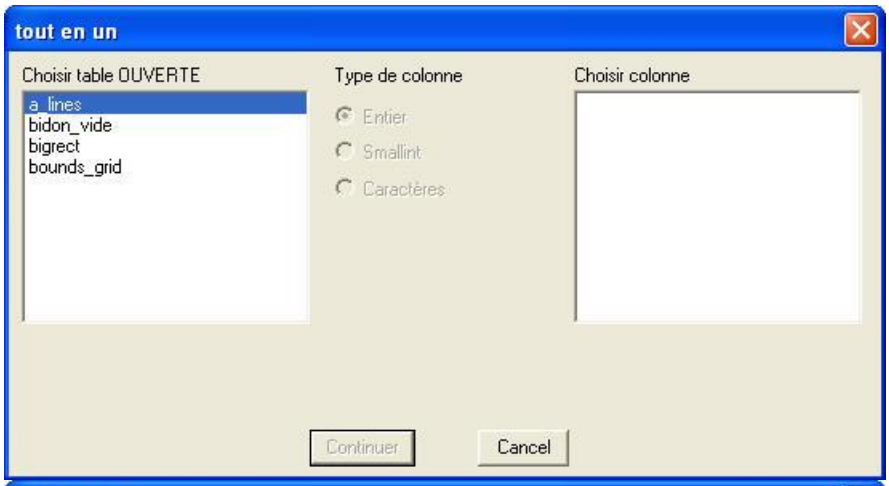


Étape 3 >>> Choix de la colonne

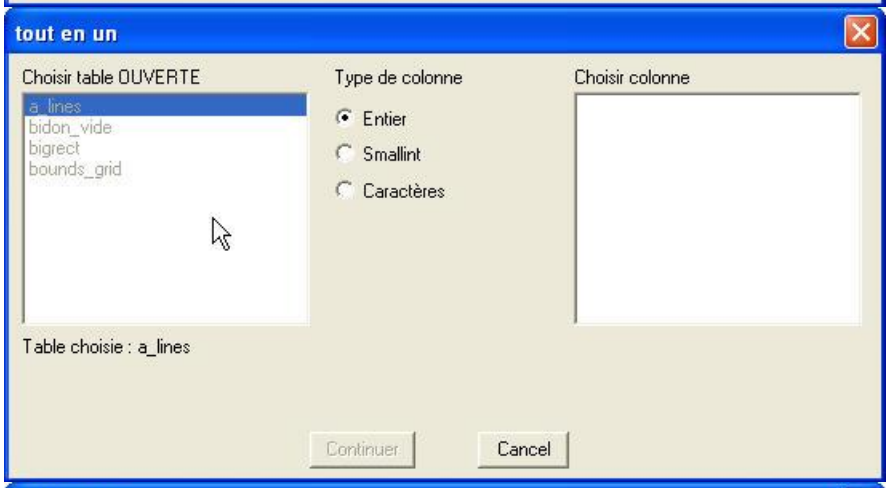
Procédure intégrée

La vérification préliminaire sur le nombre de tables ouvertes est la même. Les étapes de la définition des données sont visibles par le control qui est actif. Le bouton « Continuer » ne devient accessible que quand tout est acceptable. « Cancel » permet d'interrompre la procédure en tout temps.

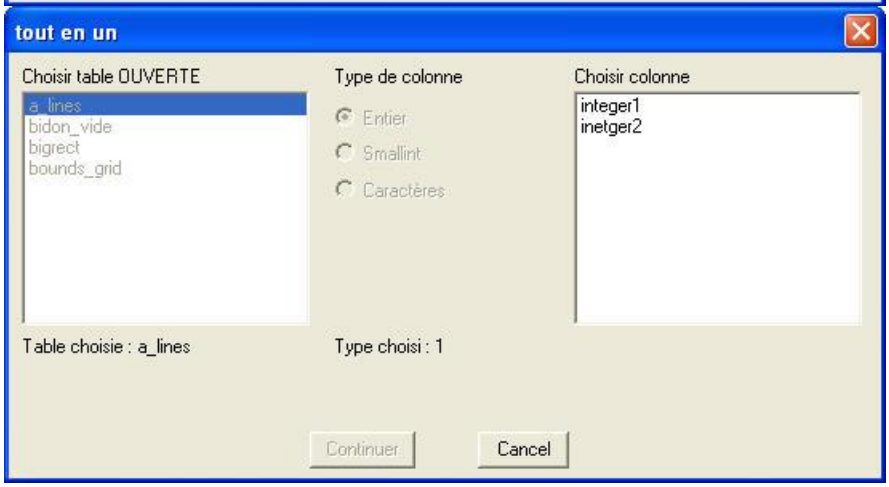
1-
Liste des tables ouvertes.
Cliquer sur votre choix



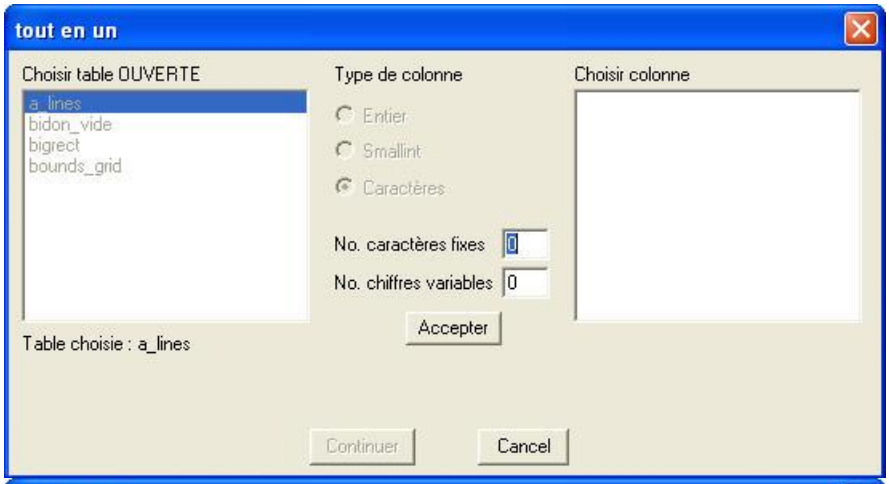
2 –
Cliquer sur le type



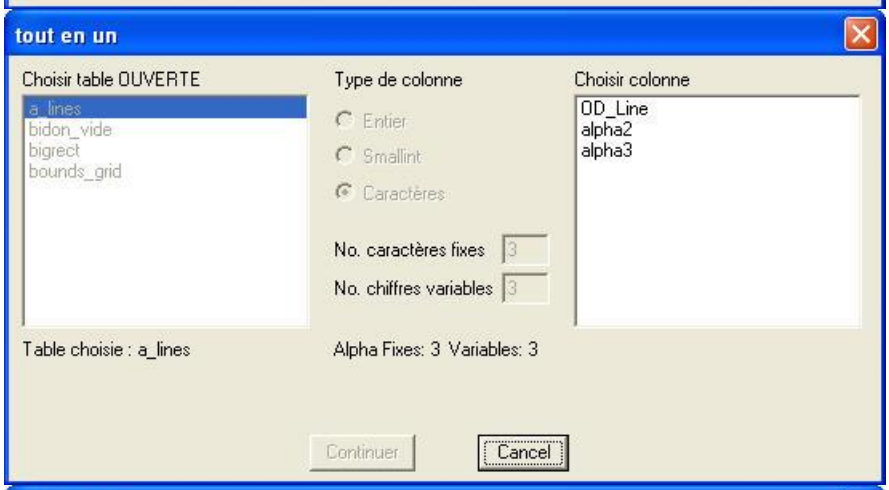
3 – a
Si type est numérique,
cliquer sur la colonne



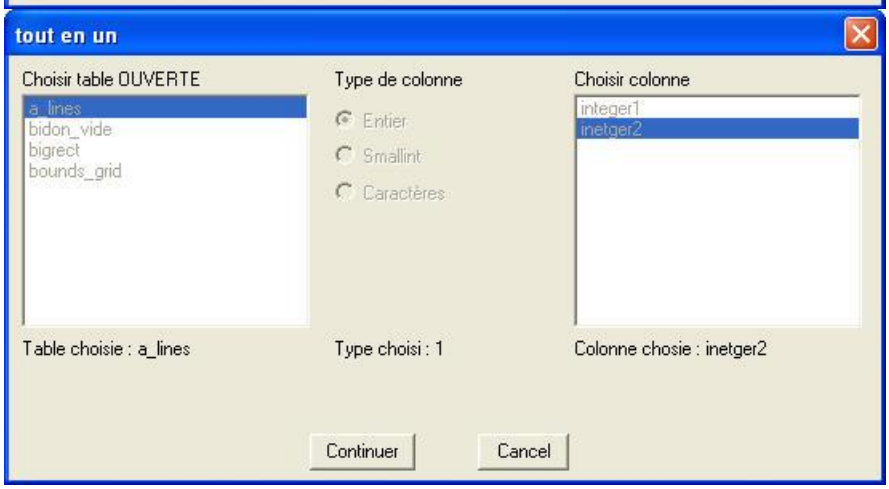
3 – b1
Si le type est Caractères, entrer les valeurs voulues puis cliquer sur « Accepter »



3 – b2
Puis choisir la colonne



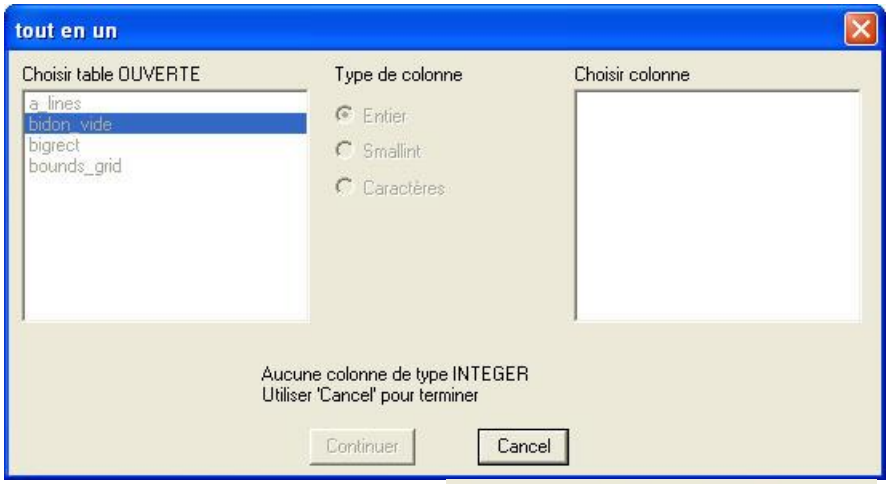
4 – a
Numérique
Tout est saisi, « Continuer » est possible.

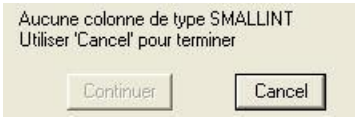


4 – b
Alpha

Le message sous « type de colonne » est conservé comme en 3 – b

Les erreurs sont signalées par des messages « intégrés ». Tous les acquis sont effacés, seul « Cancel » reste actif





MB Code pour procédure linéaire

```
Include "mapbasic.def"
Dim ltablelist, tabnames(), tabin, colnames(), colname as string
Dim NumTable, i, goal, i_typ, numcol, j, cfix, cvar as smallint
ltablelist = ""

'================== TABLE ================

NumTable = NumTables()
If NumTable <> 0 Then
      redim tabnames(numtable)
   For i = 1 To NumTable
         tabnames(i)=TableInfo( i, TAB_INFO_NAME )
      lTableList = lTableList + tabnames(i)  + ";"
    Next
  Else
    Note "Aucune table ouverte"
    Exit Sub
End If
Dialog    Title "Choisir une table ouverte" width 140
      Control ListBox      Position 10, 10    Width 120 Height 90
         Title lTableList    Value 1    into goal
      control statictext title "Sélectionner puis OK" position 10,110
      control okbutton position 90,110
if not commandinfo(1) then exit sub end if
tabin = tabnames(goal)

'================== TYPE DE COLONNE =========

Dialog title "Type de la colonne ID"
      control statictext title "Choisissez un type de colonne pour les identificateurs" position 5,5
      control radiogroup position 25,20 title "Entier;Smallint;Caractères" value 1 into i_typ
      control OKbutton
      Control CancelButton
if not commandinfo(1) then exit sub end if

'================== COLONNE ================

numcol=NumCols(Tabin)
j=0
do case i_typ
case 1
      For I=1 To NumCol
         if columninfo(tabin, "COL"+Str$(I),3)=col_type_integer then
            j=j+1
            redim colnames(j)
         colnames(j)=ColumnInfo(Tabin, "COL"+Str$(I), COL_INFO_NAME)
         end if
      next
      if j=0 then
         note "Aucune colonne de type INTEGER"
         exit sub
      end if
case 2
      For I=1 To NumCol
         if columninfo(tabin, "COL"+Str$(I),3)=col_type_smallint then
            j=j+1
```

```
            redim colnames(j)
        colnames(j)=ColumnInfo(Tabin, "COL"+Str$(I), COL_INFO_NAME)
        end if
    next
    if j=0 then
        note "Aucune colonne de type SMALLINT"
        exit sub
    end if
case 3
    dialog title "Identificateur alphanumérique"
        Control statictext title "Un identificateur alpha se compose d'une partie constante" position 5,5
        Control statictext title "(caractères ou chiffres) et d'une partie variable (chiffres)" position 5,15
        control statictext title "Nombre de caractères fixes" position 5,35
        control edittext width 20 value 0 into cfix position 100,33
        control statictext title "Nombre de chiffres variables" position 5, 50
        control edittext width 20 value 0 into cvar position 100,48
        control okbutton
        control cancelbutton
    if not commandinfo(1) then exit sub end if
    For I=1 To NumCol
        if columninfo(tabin, "COL"+Str$(I),3)=col_type_char then
            if columninfo(tabin, "COL"+Str$(I),4)>=cfix+cvar then
                j=j+1
                redim colnames(j)
            colnames(j)=ColumnInfo(Tabin, "COL"+Str$(I), COL_INFO_NAME)
            end if
        end if
    next
    if j=0 then
        note "Aucune colonne de type CHAR de largeur suffisante"
        exit sub
    end if
end case
ltablelist = ""
   For i = 1 To j
    lTableList = lTableList + colnames(i) + ";"
   Next
Dialog   Title "Choisir la colonne des Identificateurs"    width 140
     Control ListBox Position 10, 10 Width 120 Height 90 Title lTableList Value 1 into goal
     control statictext title "Sélectionner puis OK" position 10,110
     control okbutton position 90,110
if not commandinfo(1) then exit sub end if
colname=colnames(goal)

'============== RÉSULTATS ==========

Note     "Table choisie  "+tabin+chr$(13)+
         "type identif.  " +str$(I_typ)+chr$(13)+
         "Nom de Colonne "+colname
```

MB Code pour procédure intégrée

```
Include "mapbasic.def"

Declare sub main
Declare sub Tabl
Declare sub Typ
Declare sub Alpha
Declare sub finale
Declare sub lasuite

Dim ltablelist, lcollist, tabnames(), tabin, colnames(), colname as string
Dim NumTable, i, goal, i_typ, numcol, j, cfix, cvar as smallint

'============================================================
sub main
'============================================================
ltablelist = ""
lcollist=""
'================== Vérification préliminaire ================
NumTable = NumTables()
If NumTable <> 0 Then
      redim tabnames(numtable)
   For i = 1 To NumTable
            tabnames(i)=TableInfo( i, TAB_INFO_NAME )
      lTableList = lTableList + tabnames(i)  + ";"
    Next
  Else
    Note "Aucune table ouverte"
    Exit Sub
End If
'=========================================================
dialog title "tout en un" width 363 height 163

      Control statictext title "Choisir table OUVERTE" position 5,5
      Control ListBox Position 5, 15 Width 120 Height 90 Title lTableList Value 1 into goal calling TABL ID 610
      control statictext title"                                    " position 5, 108 ID 611

      control statictext title "Type de colonne" position 135,5
      control radiogroup position 135,20 title "Entier;Smallint;Caractères" value 1 into i_typ ID 620 disable calling
TYP
      control statictext title"                                    " position 135, 108 ID 621

      control statictext title "No. caractères fixes" position 135,70 ID 631 hide
      control edittext width 20 value 0 into cfix position 205,68 ID 632 hide
      control statictext title "No. chiffres variables" position 135, 85 ID 633 hide
      control edittext width 20 value 0 into cvar position 205,83 ID 634 hide
      control button width 40 height 13 title "Accepter" position 165,100 calling alpha hide ID 635

      control statictext position 105,120 ID 661 width 263 height 20 hide

      Control statictext title "Choisir colonne" position 235,5
      Control ListBox Position 235, 15 Width 120 Height 90 Title lColList Value 1 into goal ID 641 disable calling
finale
      control statictext title "                                   " position 235,108 ID 642

      control okbutton title "Continuer" position 125, 145 ID 651 calling lasuite disable
      control cancelbutton position 195, 145 ID 652
```

```
end sub
'================================================================
sub tabl
'================================================================
tabin=tabnames(readcontrolvalue(610))
alter control 610 disable
alter control 611 title "Table choisie : "+tabin
alter control 620 enable
end sub
'================================================================
sub typ
'================================================================
i_typ=readcontrolvalue(620)
alter control 620 disable
alter control 621 title "Type choisi : "+str$(i_typ)
lTableList = ""
numcol=NumCols(Tabin)
j=0
do case i_typ
case 1
      For I=1 To NumCol
            if columninfo(tabin, "COL"+Str$(I),3)=col_type_integer then
                  j=j+1
                  redim colnames(j)
            colnames(j)=ColumnInfo(Tabin, "COL"+Str$(I), COL_INFO_NAME)
            lColList = lColList + colnames(j)  + ";"
            end if
      next
      if j=0 then
            alter control 611 hide
            alter control 621 hide
            alter control 661 title "Aucune colonne de type INTEGER" +chr$(13)+"Utiliser 'Cancel' pour terminer"
show
            exit sub
      end if
      alter control 641 title lcollist enable
case 2
      For I=1 To NumCol
            if columninfo(tabin, "COL"+Str$(I),3)=col_type_smallint then
                  j=j+1
                  redim colnames(j)
            colnames(j)=ColumnInfo(Tabin, "COL"+Str$(I), COL_INFO_NAME)
            lColList = lColList + colnames(j)  + ";"
            end if
      next
      if j=0 then
            alter control 611 hide
            alter control 621 hide
            alter control 661 title "Aucune colonne de type SMALLINT" +chr$(13)+"Utiliser 'Cancel' pour terminer"
show
            exit sub
      end if
      alter control 641 title lcollist enable
case 3
      alter control 621 hide
      alter control 631 show
      alter control 632 show active
      alter control 633 show
      alter control 634 show
```

```
      alter control 635 show
end case
end sub
'====================================
sub alpha
'====================================
cfix=readcontrolvalue(632)
      alter control 632 disable
cvar=readcontrolvalue(634)
      alter control 634 disable
      alter control 635 hide
      alter control 621 title "Alpha Fixes: "+str$(cfix)+"  Variables: "+str$(cvar) show
      For I=1 To NumCol
            if columninfo(tabin, "COL"+Str$(I),3)=col_type_char then
                  if columninfo(tabin, "COL"+Str$(I),4)>=cfix+cvar then
                        j=j+1
                        redim colnames(j)
                  colnames(j)=ColumnInfo(Tabin, "COL"+Str$(I), COL_INFO_NAME)
                  lColList = lColList + colnames(j)  + ";"
                  end if
            end if
      next
      if j=0 then
            alter control 611 hide
            alter control 621 hide
            alter control 661 title "Aucune colonne de type CHAR de largeur suffisante" +chr$(13)+"Utiliser 'Cancel'
pour terminer" show
            exit sub
      end if
      alter control 641 title lcollist enable
end sub
'====================================
sub finale
'====================================
colname=colnames(readcontrolvalue(641))
alter control 641 disable
alter control 642 title "Colonne chosie : "+colname show
alter control 651 enable
end sub
'====================================
sub lasuite
'====================================
note "Données recueillies:" +chr$(13)+"   table    : "+ tabin +chr$(13)+"   colonne: "+colname  +chr$(13)+"   type
: "+str$(i_typ)
end sub
```

Changements dynamiques des contrôles

Si l'on veut réaliser un changement dans les contrôles d'un dialogue, on peut rencontrer des exigences différentes; toutes les solutions font appel à un « Alter Control » mais ce que l'on peut changer diffère selon les circonstances (voir le tableau p. 50). Nous allons prendre plusieurs exemples après avoir examiné quelques sujets généraux.

ID des contrôles

Comme un « Alter Control » ne peut marcher qu'avec le numéro ID du contrôle en question, tout contrôle qui risque d'être altéré doit recevoir un ID.

Une petite remarque sur le choix de ces ID : il ne peut pas y avoir deux contrôles avec le même ID dans le même dialogue, mais le même ID peut se retrouver dans plusieurs dialogues (il ne peut y avoir en effet qu'un seul dialogue « actif » en même temps).

Saisie de la donnée entraînant un changement

Si l'action à faire dépend de la nature d'une donnée, il faut la saisir avec un ReadControlValue(ID). Par exemple, cocher/décocher un CheckBox devrait faire alterner deux RadioGroup. Nous devrons donc avoir dans la définition du CheckBox un « ID *1000* » et un « Calling *subA* » et cette routine aurait la forme générale suivante

```
Subroutine SubA
If readcontrolvalue(1000) then
        'faire ce qu'il faut si le control vient d'être coché
  else
        'faire ce qu'il faut si le control vient d'être décoché
end if
end sub
```

Changement de contenu ou de contenant

Les changements que l'on peut apporter portent sur le contenu (un nouveau message dans un StaticText) ou sur le contenant (les contrôles eux-mêmes).

Le contenu est modifié en donnant dans un Alter Control une nouvelle valeur à TITLE ou VALUE. Nous avons vu plusieurs principes et exemples des opérations en particulier dans Dimensionnement initial avant un Alter Control (p. 8), Maintien des valeurs saisies (p. 9), Procédure intégrée (p. 36). Dans tous les cas, les changements portent sur un contrôle bien défini.

À la charnière entre contenu et contant se trouverait un texte qui apparaît dans certaines circonstances, car il y a deux façons de le faire apparaître. Il peut être créé dans le dialogue avec son contenu et caché jusqu'au moment propice (ce serait le cas en particulier d'un message unique et fixe), ou le Control StaticText est vide lors de sa

création et son contenu TITLE est modifié par la suite (le cas de messages multiples situationnels)

Le fait qu'un contrôle devienne accessible ou interdit d'accès (ENABLE, DISABLE) touche déjà le domaine du contenant car un contrôle inaccessible est comme s'il était caché (HIDE, SHOW). Dans le montage de la Procédure intégrée (p. 36), un groupe de contrôles n'apparaissent que dans certaines conditions.

Tous ces changements sont soumis à une contrainte fondamentale, celle de l'occupation de l'espace interne de la fenêtre d'un dialogue. Chaque contrôle doit occuper son propre espace; empiéter sur un autre aboutirait à des occultations désagréables s'il ne s'agit que de textes, catastrophiques si ce sont des zones de saisie de données. Il faut donc imaginer un dialogue comme un espace compartimenté, chaque élément étant le domaine exclusif d'un contrôle.

Contrôles superposés ou étalés

Mais il en est avec les contrôles comme avec les dialogues. Plusieurs dialogues peuvent être ouverts en même temps (Carte et tableau intégrés, p.28) mais seul le dernier créé est actif. De même plusieurs contrôles existent en même temps mais seuls les visibles (SHOW) peuvent être actifs. Il est donc tout à fait possible d'avoir deux ou plusieurs contrôles définis pour le même compartiment dont un seul peut être visible, les autres restant cachés.

On peut se demander quelle solution adopter : devrait-on afficher tous les contrôles existant côte-à-côte seuls certains étant accessibles, ou ne montrer que ceux qui seraient accessibles à un moment donné dans des compartiments en nombre réduit? Je crois que tout dépend du nombre de contrôles par rapport au nombre de compartiments. Plus la densité est élevée, plus le recours aux contrôles superposés devient la solution préférée. Cela limite la taille des dialogues et ne met pas l'utilisateur en présence d'une information volumineuse et souvent inutile. L'exemple suivant donne une idée des deux solutions même si la taille du dialogue est petite.

```
declare sub change
dim ilog as logical

dialog title "Test de controls superposés"
control checkbox position 10,10 title "Avec le nord en haut" value 0 into ilog calling change ID 1000
control radiogroup position 10,30 title "Paysage (romantique);portrait (fidèle)" ID 1010
control radiogroup position 10,30 title "bleu ciel;rouge sang" ID 1011 hide
control okbutton

sub change
ilog=readcontrolvalue(1000)
do case ilog
case 0
alter control 1010 show
alter control 1011 hide
case 1
alter control 1010 hide
alter control 1011 show
end case
end sub
```

À l'ouverture

Après décochage

À l'ouverture

Après décochage

```
declare sub change
dim ilog as logical

dialog title "Test de contrôles désactivés"
control checkbox position 35,10 title "Avec le nord en haut" value 0 into ilog calling change ID 1000
control radiogroup position 10,30 title "Paysage (romantique);portrait (fidèle)" ID 1010
control radiogroup position 100,30 title "bleu ciel;rouge sang" ID 1011 disable
control okbutton

sub change
ilog=readcontrolvalue(1000)
do case ilog
case 0
alter control 1010 enable
alter control 1011 disable
case 1
alter control 1010 disable
alter control 1011 enable
end case
end sub
```

« Alter Control » : Actions possibles

La syntaxe de cet énoncé est simple, elle exige un *ID* (un nombre, sans mot-clé le précédant) et au moins un des mots-clés suivants (avec, selon le cas, une donnée pertinente); toute combinaison de mots-clés acceptables est possible avec un seul énoncé.

Voir les détails des conditions dans les tableaux des mots-clés des « Contrôles » (p. 5)

	Title *nouvelle_phrase*	Title From Variable *nu_var*	Value *nu_valeur*
Statictext	oui	non	non
Edittext	oui	non	phrase
Listbox	oui, liste d'items	Var. carac. dim.	# item sélectionné
Multilistbox	oui, liste d'items	Var. carac. dim.	# 1er item sélectionné
Popupmenu	oui, liste d'items	Var. carac. dim.	# item sélectionné
Checkbox	oui	non	1 pour coché, 0 pour pas coché
Radiogroup	oui, liste d'items	Var. carac. dim.	# item sélectionné
Buttons (all)	oui	non	non
Style picker (tous)	non	non	Makexxx(…) ou *var_xxx*
DocumentWindow	non	non	non
GroupBox	oui	non	non

Disable = met le control en grisé (il n'est plus accessible)
Hide = le control n'est plus affiché
Active = le control est en focus

	Enable or Disable	Show or Hide	Active
Statictext	non	oui	non
Edittext	oui	oui	oui (*)
Listbox	oui	oui	non
Multilistbox	oui	oui	non
Popupmenu	oui	non (**)	non
Checkbox	oui	oui	non
Radiogroup	oui	oui	non
Buttons (all)	oui	oui	non
Style picker (tous)	oui	oui	non
DocumentWindow	oui	oui	non
GroupBox	non	oui	non

(*) Permet d'entrer directement du texte dans cet edittext. Pourrait être inclus dans le dialog_handler si l'entrée du texte était la première action à faire

(**) Apparemment le seul contrôle qui ne peut pas être caché selon l'Aide MB. En fait il peut l'être.

Fermeture d'un dialogue

Il y a plusieurs façons de fermer un dialogue, certaines résultent de l'intervention de l'utilisateur, d'autres sont programmées. Nous avons déjà rencontré plusieurs types d'intervention; ce sont les boutons OKButton et CancelButton et la petite croix à droite dans la barre titre du dialogue, quand elle existe. La fermeture peut être programmée par un « Dialog Remove » exécuté dans un control_handler du dialogue. On peut vouloir que l'utilisateur confirme la perte d'information contenue dans un dialogue s'il a utilisé CancelButton; « Dialog Preserve » restaure alors le dialogue.

OKButton assigne toutes les valeurs définies dans le dialogue aux variables INTO.

CancelButton ignore toute assignation

Croix de barre titre n'existe que si le dialogue a un TITLE, ne marche que s'il y a un contrôle autre que OKButton dans le dialogue. Ignore toute assignation

Détection des conditions de fermeture : « CommandInfo() »

On peut repérer une condition de fermeture d'un dialogue en utilisant juste après le code qui définit le dialogue l'énoncé « Commandinfo(1) » qui est VRAI si le bouton OKButton a été utilisé.

```
Dialog …
        …
        Control OKButton
        Control CancelButton
If not CommandInfo(1) then
        'fermeture par autre moyen que OKButton; suite, par exemple >
        exit sub
end if
'suite des opérations
```

Commandinfo(1) = VRAI permet de savoir que l'utilisateur a validé le dialogue et que les opérations peuvent continuer, mais s'il est FAUX, il n'y a pas moyen de savoir dans quelles circonstances, à quel endroit dans le dialogue, la fermeture a pris place. Cela peut créer des problèmes par la suite surtout quand des opérations ont pu déjà être faites suite à certains choix exprimés par l'utilisateur (via des control_handlers) et que certains résultats de ces opérations doivent être effacés avant de continuer (ex. fenêtre ouverte qu'il ne faudrait pas conserver).

Une procédure linéaire a cet avantage de savoir exactement quand une interruption a lieu et de prendre les mesures en conséquence. Il faudrait ajouter dans l'approche intégrée un indicateur de position (variable en commun) mis à jour par chaque control_handler qui permettrait de localiser l'interruption après la fermeture du dialogue; dans le code ci-dessus, les opérations de nettoyage se feraient alors avant le « Exit Sub »

Par exemple, on dimensionne en commun la variable i_stop comme smallint et on l'initiale dès le début du programme à 1. Ensuite dans chaque handler, dès qu'une opération clé a été faite (par exemple dès qu'une table est ouverte) on met à jour i_stop par la valeur suivante. Ainsi on peut savoir par cette variable quelles sont les opérations qui ont été faites avant la fermeture « non-conforme » du dialogue et les « défaire » (par exemple fermer une table).

Fermeture programmée : « Dialog Remove »

Cette commande ne peut être lancée que depuis un control_handler; elle s'applique donc au dialogue actif, le dernier créé. Le dialogue ne disparaît pas instantanément; ce n'est que lorsque la procédure du control_handler aura terminé que le dialogue sera fermé.

```
declare sub ferme

dialog Title "Double remove"
    control button title "fermer dialogue" calling ferme
    control okbutton

sub ferme
    dialog remove
    note "La commande est passée; le dialogue sera retiré après avoir fermé ce message"
end sub
```

Le message a été déplacé pour montrer que le dialogue existe encore

Restauration après OK/CancelButton : « Dialog Preserve »

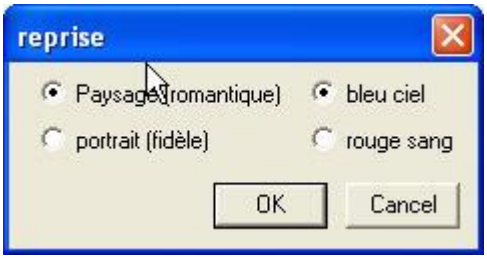
Cette commande ne peut être placée que dans un handler d'un bouton OKButton ou CancelButton. Elle offre à l'utilisateur la possibilité de revoir ses choix qu'il avait acceptés (OK), ou de refaire des choix s'il avait cancellé une première fois (Cancel).

```
declare sub reprend

dialog Title "reprise"
    control radiogroup position 10,5 title "Paysage (romantique);portrait (fidèle)"
    control radiogroup position 100,5 title "bleu ciel;rouge sang"
    control okbutton
```

```
    control cancelbutton calling reprend

sub reprend
if ask("Voulez-vous abandonner ou reprendre","Abandonner","Reprendre")=0 then
    dialog preserve
end if
end sub
```



Dialogue initial

Dialogue avant « Cancel »

Après « cancel », le dialogue est fermé
Si « Reprendre », le dialogue est affiché
à nouveau dans l'état d'avant « cancel »

Un dialogue toujours au coin

Le problème que je me suis posé est de trouver un moyen de conserver un dialogue dans le coin inférieur droit de la fenêtre MapInfo, quelle que soit la taille de la fenêtre.

Je me suis heurté à de nombreux problèmes techniques parce que l'information nécessaire pour réaliser cela fait appel dans MI à plusieurs systèmes de référence (origine et unité de mesure). Passons-les en revue d'abord.

Les données sur les fenêtres (comme la fenêtre MapInfo, code 1011) que l'on peut extraire avec WindowInfo(1011,4) pour la largeur et …,5) pour la hauteur, sont en unités papier, qui sont (ma traduction) « *interne et invisible de l'utilisateur. Quand un utilisateur fait une opération qui affiche une mesure-papier*[2]*, l'unité de mesure affichée à l'écran est indépendant de l'unité papier interne de MapBasic* »[3] . Par défaut, l'unité est le pouce et nous allons la conserver.

Un dialogue fait appel à deux systèmes. Les dimensions dans un dialogue sont exprimées en unités dialogue UD (largeur, hauteur du dialogue et des contrôles, positions x et y des contrôles dans le dialogue). Par contre la position (x et y) du dialogue dans la fenêtre MapInfo est exprimée en pixels.

Ma première expérience a été de trouver une façon de fixer la largeur et la hauteur d'un dialogue pour couvrir exactement la fenêtre MapInfo. Noter que les paramètres numériques que j'ai adoptés peuvent être différents sur d'autres installations et pourraient devoir être recalibrés.

Largeur : le plus rapide, la conversion fut vite établie de la façon suivante

```
dialog  title "position" width windowinfo(1011,4)*64 - 6 height 40 position 0,0
```

Le facteur de 64 convertit les pouces-papier en UD; 6 représente probablement l'épaisseur constante des bordures de boîte.

Hauteur : conversion plus délicate qui a commencé par trouver une valeur constante (0.56 pouce) à enlever de la hauteur donnée, incluant à en particulier la barre de message du bas de la fenêtre MI. Le facteur de conversion est passé à 59.

```
dialog  title "position" width 40 height windowinfo((1011,5) - 0.56)*59 position 0,0
```

Le dialog construit avec ces deux expressions couvre bien toute la fenêtre MI à toutes les tailles utiles possibles.

Comme la largeur et la hauteur ainsi calculées sont assimilables aux Xmax et Ymax de la fenêtre MI en UD, nous pouvons nous en servir pour positionner le dialogue en pixels. Encore une fois une très courte expérience m'a montré que le facteur de conversion

[2] Quelle que soit la signification possible de cette expression
[3] Guide de référence de MapBasic, sous "Set Paper Units", page 542 de la version 6.5

entre UD et pixels était de 1.5 pour la largeur, mais pas pour la hauteur. J'ai supposé qu'il devrait être dans les mêmes proportions que les facteurs inventés dans la première phase, et en effet le facteur pour la hauteur de 1.627 convient très bien.

J'ai donc pu écrire le code qui répond au problème que je m'étais posé. J'ai voulu mettre en évidence les paramètres du problème (*hei* hauteur, *wid* largeur) et le calcul intermédiaire des dx et dy de la position du dialogue pour faciliter la lecture mais dans la pratique tout pourrait être condensé dans l'expression Dialog sans avoir besoin de variables intermédiaires.

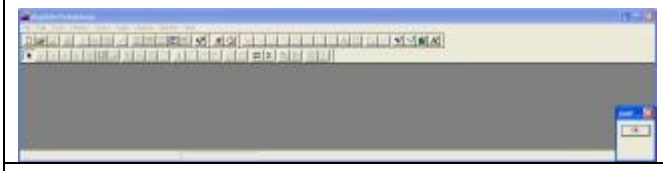
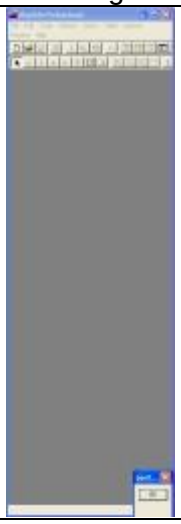
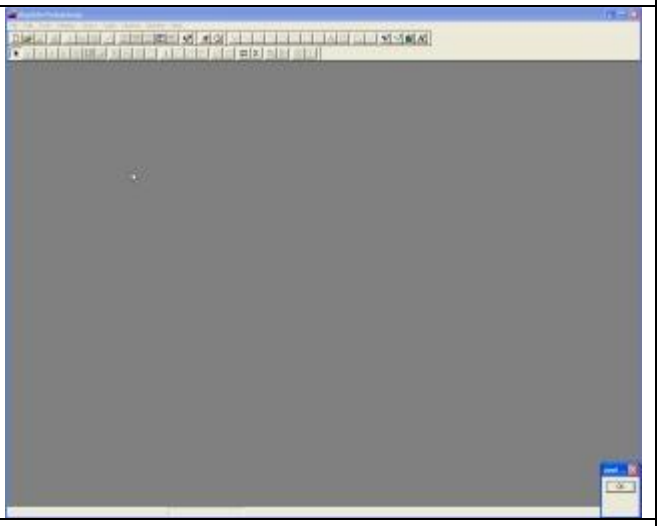
```
dim hei, wid, dx, dy as float

hei=50
wid=50

dx=(windowinfo(1011,4)*64-6-wid)*1.5
dy=((windowinfo(1011,5)-.56)*59 -hei)*1.627

dialog title "position" Width wid Height hei Position dx,dy
control okbutton
```

Il est bien entendu que ceci n'est possible que pour un dialogue dimensionné explicitement.

	
Pleine largeur	Moitié-moitié
	
Pleine hauteur	Plein écran

BMP et custom symbols

Nous avons déjà vu certains des éléments essentiels de l'utilisation d'un « custom symbol » comme logo dans un dialogue. Cette partie porte plus sur des aspects techniques de ces BMP et de leur mise en œuvre.

Tout d'abord il faut savoir que la taille maximale d'un symbole est de 48 « points » quelle que soit la taille du BMP qui lui sert de base. Une petite exploration sur ma machine m'a montré que ce symbole affiché est une image de 64x64 pixels. Il y a en fait une assez bonne relation entre points et pixels pour que la formule « 16*(points)/12 arrondi à l'entier le plus proche » marche assez bien. Mais attention aux plus petites tailles (<4 pts).

Avoir un BMP de plus de 64x64 n'apporte donc aucun avantage supplémentaire dans la précision, la netteté ou la qualité des couleurs dans toute la gamme des tailles en points des symboles affichés en particulier pour leur taille maximale. Un BMP de 64x64 pèse 8 ko environ; le BMP maximum correspond à une image de 511x511 pixels et pèse 128 ko (maximum pour la version 7.0, a pu être augmenté depuis). Cette inflation est totalement inutile à moins que l'on ne veuille utiliser à moindre coût une image existante sans la manipuler un peu auparavant.

Comme il faut toujours considérer aussi l'utilisation normale des symboles, répondre à la question « Y a-t-il une meilleure taille pour les BMP ? » entraîne à chercher une solution tenant compte des gammes de variations du BMP (en deçà de 8x8 pixels on ne peut guère avoir que des formes très simples, au-delà de 64x64 aucun détail supplémentaire ne serait perçu) et du symbole affiché (en deçà de 6 points peu de détails vont être visibles, et il est impossible de dépasser les 48 points).

Le compromis est conditionné par le besoin de cartographier une trame BMP sur une trame symbole les deux trames n'ayant pas forcément la même taille. Si le BMP est petit (ex. 8x8) et la taille grande (ex. 32), le symbole affiché montrera une très forte pixellisation (diagonales en escaliers). Si le BMP est grand (ex 64x64) et la taille faible (ex. 24), il va y avoir une condensation des pixels qui entraîne ce qui serait une assignation un peu au hasard des couleurs, donc une destruction de l'intégrité de l'image.

Une bonne solution pratique est probablement celle de MapInfo qui a choisi de faire des BMP de 32x32, mais s'il s'agit de l'affichage d'un logo, je recommanderais une taille de 64x64.

Utiliser des BMP comme logos dans des dialogues exige que le fichier BMP soit installé dans le répertoire dans lequel MI va aller le chercher et qu'il soit accessible. Comme un tel BMP n'est pas forcément déjà dans le répertoire en question, il faut donc que l'application le détecte et au besoin l'installe. Une bonne pratique est de toujours distribuer ce BMP avec l'application et de demander à ce que ces deux fichiers soient placés dans le même répertoire. Voici un code MB pour le faire. Remarquez le choix à

faire entre les deux approches (dure, l'application prend fin, - douce, un simple avertissement)

```
if not fileexists(locatefile$(8)+"testicon_64.bmp") then
    if not fileexists(applicationdirectory$()+"testicon_64.bmp") then
        note "Attention, le fichier 'testicon_64.bmp' n'est pas dans le répertoire de l'application"
            +" ni dans le répertoire CUSTSYMB"+chr$(10)+chr$(10)+
'Approche dure 2 lignes "commented out" par apostrophe
'           "L'application ne pourra pas être lancée"
'           end program
            "L'application sera lancée sans le logo de la compagnie XYZ"
      else
        save file applicationdirectory$()+"testicon_64.bmp" as locatefile$(8)+"testicon_64.bmp"
        reload custom symbols from locatefile$(8)
    end if
end if

dialog Title "custom symbol"
    control symbolpicker value makecustomsymbol("testicon_64.bmp",0,48,0) width 48 height 44
    control okbutton
```

Remarquez aussi la définition de la taille du SymbolPicker; il y a 4 UD de plus en largeur qu'en hauteur. Avec de telles dimensions, le BMP est affiché comme un carré de 64x64 pixels, entouré d'un cadre inamovible et ineffaçable de 4 pixels de large (1 blanc, 1 noir, 2 arrière-fond pour haut et gauche, le blanc remplacé par un autre noir pour droite et bas).

Finalement, toute cette approche risque d'entraîner des difficultés sur les installations offrant aux utilisateurs des droits d'accès limités aux répertoires « publics » comme pourrait l'être le CUSTSYMB de Mapinfo. Cet inconvénient n'existe pas avec la solution « DocumentWindow » car l'image à afficher peut rester dans le répertoire de l'application.