Visual Basic :quelques trucs en vrac

Si vous débutez en Visual Basic, vous avez du vous demander comment écrit-on ceci. Si vous utilisez l'aide en ligne, comment le formuler. Et si vous avez trouvé, là, il vous sort 150 lignes de titre. C'est la raison de ce petit aide mémoire.

Note : Police non proportionnelle = courrier

Form Feuille	TexBox (Boite Texte)	Combo et List Box listes simples & combinées	Les Variables	Les Boucles	CommandButton Bouton de commande
Fichier Texte	Fichier, accès direct	Afficher une aide simple	Help WorkShop	Les tests	Menus et menus contextuels
Objets basiques	Presse papier	Boites de dialogues standards	MsgBox	Option Bouton	Barre d'outils & la barre d'état
Valeur des touches	Tableaux	Les imprimantes	Contrôle de saisie	SetFocus : bug	
Base SQL Server	Base Access	ADO Data Control	Modèle ADO (code)		

Convention d'écriture des objets

Frm = Form (feuille)	Menu = Mnu	PictureBox = Pic	Boite Texte = Txt (TextBox)	Label = Lbl
Bouton de commande = Cmd	Cadre = Fra (Frame)	Bouton d'option = Opt	Case à cocher = Chk (CheckBox)	Bouton d'option = Opt
Liste Modifiable = Cbo (ComboBox)	List simple = Lst (List Box)	Ascenseur Horiz. = Hsb	Ascenseur Vertical = Vsb	Minuterie = Tmr (Timer)
Liste unités fixes =	Liste répertoires =	Liste fichiers = Fil	Formes = Shp	Lignes = Lin

Drv (DriveBox)	Dir (DirListBox)	(FileListBox)	(Shapes)	(Lines)
Dessin = Img (Image)	Contrôle de données = Dat (Data)	OLE = Ole	Onglet = Ong	DBlist = Dbg
Grille Base de Données = Dbg (DbDrid)	Liste Modifiable Base de Données = Dbc (DbCombo)			

Les Form ou feuilles

- ControlBox False cache les contrôles (de fermeture, agrandissement ...) de la feuille (Form)
- Sub Form_load est utilisé pour le code avant chargement de la feuille.
- WindowStade => Maximized affiche la feuille plein écran
- caption = texte qui sera affiché dans la barre des titres.

Les TextBox

Propriétés modifiables non (?) dans le code :

- Multiline => True, pour afficher les sauts de lignes
- ScrollBars (0 - none => aucunes ; 1 - Horizontal ; 2 - Vertical ; 3 - Both => les 2), pour les ascenseurs

Propriétés modifiables dans le code :

- .SelStart sélection position début
- .SelLen sélection longueur

Combo/List Box

ListBox

Il existe des objets similaire : DriveListBox, DirListBox, FileListBox

Evènements (le focus : l'objet est en court, sélectionné) :

_KeyPress appuie d'1 touche	lorsque l'utilisateur enfonce (_KeyDown) ou relâche (_KeyUp) une touche tandis qu'un objet à le	L'objet reçoit _GoFocus ou pert (_LostFocus) le focus	_DropDown la liste va se dérouler	_Change

	focus.			
_Validate(KeepFocus As Boolean)				

.ItemChek Si liste en case à cocher (style=1)

Propriétés

.ListCount nb éléments (0->n)	.List (x) élt indexé x	.NewIndex Index dernier ajouté	.index N° sélectionné (pdt controle)	.Selected(index) [= boolean]
.clear Efface la liste	.AddItem Ajoute 1 élément	.RemoveItem Supprime 1'élément		

Note :

on peut tester si .ListIndex <> -1 pour savoir s'il y a un élément sélectionné (au lieu de .selected(index)). De même on peut mettre .ListIndex = -1 au lieu .selected(index) = False . Cela, bien sûr n'est pas tout à fait valable en cas de multi-sélection.

.ItemData (long) = associe un tableau de nombres au tableau list (espèce de fichier index). Ex:

```
Lst.AddItem "Lucie Gombaud"                                    ' prénom & nom
Lst.ItemData(Lst.NewIndex) = 42310                             ' matricule
```

Remarque :

- pour trier, vous pouvez affectez la valeur True à la propriété Sorted du contrôle ListBox (ne fonctionne qu'à l'initialisation de la feuille).
- si on affecte la valeur True à la propriété Sorted du contrôle ListBox, cela la trie (ne fonctionne pas dans le code même en form_load). Il faut parfois actualiser (F5).

ComboBox

Même propriétés et évènements que les ListBox, plus quelques uns spécifique :

Style => DropDown List

Valeur des touches clavier

vbKeyBack = Touche RET.ARR (8)	vbKeyReturn = Touche ENTRÉE (13)

Les valeurs entre parenthèses sont en Hexa. Pour plus de détail sur d'autres touches voir dans

l'aide VB "Constantes de code de touches".

Les Tableaux

ex : Dim Tbl(100) As Integer

Redim

Déclarer le tableau sans taille Dim Tbl() As Integer, faire un redim de l'estimé Redim Tbl(100),
puis un redim après comptage Redim Tbl(x)

Dim Rep As string * 1

Les Variables

Byte	1 octet	0 à 255
Boolean	2 octets	True ou False
Integer	2 octets	-32 768 à 32 767
Long (entier long)	4 octets	-2 147 483 648 à 2 147 483 647
Single (virgule flottante, simple précision)	4 octets	-3,402823E38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E38 pour les valeurs positives
Double (à virgule flottante en double précision)	8 octets	-1,79769313486232E308 à -4,94065645841247E-324 pour les valeurs négatives ; 4,94065645841247E-324 à 1,79769313486232E308 pour les valeurs positives
Currency (entier à décalage)	8 octets	-922 337 203 685 477,5808 à 922 337 203 685 477,5807 Decimal 14 octets +/-79 228 162 514 264 337 593 543 950 335 sans séparateur décimal
Date	8 octets	1er janvier 100 au 31 décembre 9999
Object	4 octets	Toute référence à des données de type Object
Variant (nombres)	16 octets	Toute valeur numérique, avec la même plage de valeurs qu'une donnée de type Double

Variant (caractères)	22 octets	+ longueur de la chaîne Même plage de valeurs qu'une donnée de type String de longueur variable

Option Explicit

Cette ligne en début de programme vous obligera à déclarer vos variables avant de les utiliser.

Vous pouvez généraliser cela pour tous vos programmes automatiquement : Outils/Options/Editeur cocher "Déclaration des variables obligatoire" (dans Paramètres du code)

Opération sur les zones alphabétiques

Right(zone,x)	prend les **x** caractères à partir de la droite
Left(zone,x)	prend les **x** caractères à partir de la gauche
Mid(zone,x,y)	prend **y** caractères à partir de **x** (si pas **y**, prend tout après **x**) attention x commence à 1
Str(zone_numérique)	Transforme une zone numérique en alpha
Val(zone_numérique)	Transforme une zone alpha en numérique
Chr(13) + Chr(10)	Génère un saut de ligne (pour test utiliser plutôt Chr(13) & Chr(10))
UCase(ZoneAlpha) LCase(ZoneAlpha)	Converti en majuscule ou minuscule

Un petit truc pour formater une ligne : Right(" " + Str(zone_de_3_chiffres), 3) A quoi çà sert, ben, pour sauvegarder/charger des nombres à partir d'un fichier texte.

Toujours pas compris ? Vous voulez gérer le nombre de billes des élèves d'une classe sans passer par un base de donnée. Donc vous avez : Paul 10 billes, Jacques 5....
Mais, vous ne pouvez pas sauvegarder additionner du texte ni avoir un texte+zone numérique dans le fichier.
Donc vous faites un fichier texte du style 25 caractères pour le prénom + 3 caractères pour le nombre de billes. Puis en jouant avec les Val et Str, les Mid, vous avez tout.

Les conversions

Fonction	Type renvoyé	Plage de valeurs de l'argument expression
CBool	Boolean	Toute chaîne ou expression numérique valide.
CByte	Byte	0 à 255.
CCur	Currency	-922 337 203 685 477,5808 à 922 337 203 685 477,5807.

CDate	Date	Toute expression de date valide.
CDbl	Double	-1,79769313486232E308 à -4,94065645841247E-324 ou 4,94065645841247E-324 à 1,79769313486232E308
CDec	Decimal	+/-79 228 162 514 264 337 593 543 950 335 sans décimales ou, +/-7,9228162514264337593543950335 à 28 décimales. Le + petit nombre <> 0 est 0,0000000000000000000000000001.
CInt	Integer	-32 768 à 32 767 ; les fractions sont arrondies.
CLng	Long	-2 147 483 648 à 2 147 483 647 ; les fractions sont arrondies.
CSng	Single	-3,402823E38 à -1,401298E-45 ou, 1,401298E-45 à 3,402823E38
CStr	String	Les valeurs renvoyées par la fonction Cstr dépendent de l'argument expression
CVar	Variant	Même plage de valeurs que le type Double pour les nombres et que le type String pour les chaînes non numériques.

Contrôle de saisie

Il existe un composant qui peut créer des masque de saisie (Miscrosoft Masked Edit Control). Il fonction un peu comme une TextBox avec des propriétés supplémentaires (Format, Mask, MaxLength, PromptChar...)

MsgBox

MsgBox(prompt[, buttons] [, title] [, helpfile, context])

Dim rep
rep = MsgBox("Fin d'application", vbOKOnly + vbExclamation)
rep = MsgBox("Fin d'application", 48) idem ci-dessus

prompt Expression de chaîne affichée comme message dans la boîte de dialogue. La longueur maximale de l'argument prompt est d'environ 1 024 caractères selon la largeur des caractères utilisés. Si l'argument prompt occupe plus d'une ligne, n'oubliez pas d'insérer un retour chariot (Chr(13)) ou un saut de ligne (Chr(10)) entre les lignes, ou une combinaison de caractères retour chariot-saut de ligne (Chr(13) & Chr(10)).

buttons Facultatif.Expression numérique qui représente la somme des valeurs indiquant le nombre et le type de boutons à afficher, le style d'icône à utiliser, l'identité du bouton par défaut, ainsi que la modalité du message. Si l'argument buttons est omis, sa valeur par défaut est 0.

title Facultatif. Expression de chaîne affichée dans la barre de titre de la boîte de dialogue. Si l'argument title est omis, le nom de l'application est placé dans la barre de titre.

Valeurs buttons Valeurs renvoyées

Constant	Value	Description
vbOKOnly	0	Affiche le bouton OK uniquement.
vbOKCancel	1	Affiche les boutons OK et Annuler.
vbAbortRetryIgnore	2	Affiche le bouton Abandonner, Réessayer et Ignorer.
vbYesNoCancel	3	Affiche les boutons Oui, Non et Annuler.
vbYesNo	4	Affiche les boutons Oui et Non.
vbRetryCancel	5	Affiche les boutons Réessayer et Annuler.
vbCritical	16	Affiche l'icône Message critique.
vbQuestion	32	Affiche l'icône Requête d'avertissement.
vbExclamation	48	Affiche l'icône Message d'avertissement.
vbInformation	64	Affiche l'icône Message d'information.
vbDefaultButton1	0	Le premier bouton est le bouton par défaut.
vbDefaultButton2	256	Le deuxième bouton est le bouton par défaut.
vbDefaultButton3	512	Le troisième bouton est le bouton par défaut.
vbDefaultButton4	768	Le quatrième bouton est le bouton par défaut.

Constante	Valeur	Description
vbOK	1	OK
vbCancel	2	Annuler
vbAbort	3	Abandonner
vbRetry	4	Réessayer
vbIgnore	5	Ignorer
vbYes	6	Oui
vbNo	7	Non

Fichier Texte

Lecture d'un fichier texte

Texte entier mis dans la variable Texte :

```
Dim CheminFichier, Texte As String
Dim a As Long
CheminFichier = "c:\Autoexec.bat"
a = FreeFile() ' pour obtenir le No de fichier disponible
Open CheminFichier For Input As #a
```

```
Texte = Input(LOF(a), a)
Close #a
```

Lecture ligne par ligne et mis dans le tableau iteme() :

```
Dim lg As String       ' fichier à ouvrir
Dim x As Integer       ' compteur
Dim iteme() As String  ' tableau

x = 1
lg = ""
Open "h:\mes documents\item\ListInvent.txt" For Input As #1
Line Input #1, lg
Do While Not EOF(1)
  Line Input #1, lg
  x = x + 1
Loop
Close #1
Redim iteme(x)         ' on pourrait mettre (x-1) mais en cas de fichier
vide ?
```

A la place du nom du fichier avec son chemin, on peut mettre une variable.
La procédure ci-dessus compte et dimensionne un tableau (dans le but de le remplir). Cela évite d'allouer trop de mémoire à un tableau (ou pas assez ☺) si on n'en connaît pas trop le nb d'éléments .

Fichier à accès direct

DEBUT

Mettre dans un module (au début, après option explicit)

```
Type EnrItem
  Num As String * 5
  Lib As String * 40
End Type
```

Puis dans la feuille

```
Dim Item As EnrItem
Open "ItemFich.txt" For Random Access Write As #1
Put #1, num_enr, Item                  ' écrit
..
Get #1, [num_enr], Item                ' Se positionne et lit
..
Seek #1 num_enr                ' Se positionne
```

Pour les fichiers binaires :

```
Open "ItemFich.txt" For Binary Access Write As #1
Get #1, [No_octet], variable
ou
Seek #1, No_enr
x = Input$(1,8)
...
Put #1, No_enr, variable
```

Les Boucles

DEBUT

Do ... [Exit Do] ...	Do While *condition* ... [Exit Do]	Do Until *condition* ... [Exit Do]	Do ... [Exit Do] ...	While *condition* ... Wend

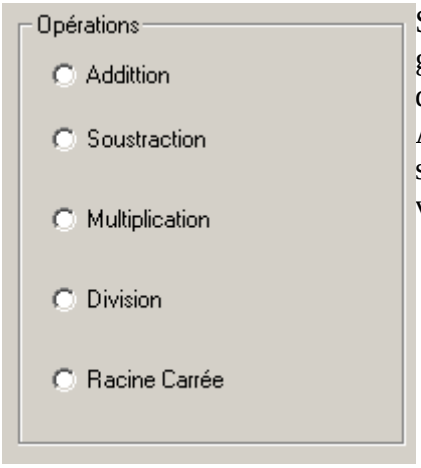
Loop	... Loop	... Loop	Loop Until *condition*	

CommandButton

Vous pouvez remplacer le texte par une image :

- Le nom externe (= **Caption**) peut être effacé.
- Propriété **Picture** choisir l'image
- Propriété **Style** choisir **1 - Graphical**

Option Bouton



Sur une feuille, les boutons d'option constituent un seul groupe. Pour créer des groupes de boutons d'option, vous devez les placer dans des contrôles Frame ou PictureBox. Ainsi, ils peuvent être sélectionnés et déplacés comme une seule unité (maintenez enfoncée la touche CTRL pendant que vous dessinez un cadre autour).

Exemple :

```
Private Sub NomProcedure_Click(Index As Integer)
Select Case Index
  Case Is = 1
    ...
  Case Is = 2
    ...
End Select
```

Création de touches de raccourci. Vous pouvez utiliser la propriété Caption pour créer des touches d'accès rapide pour vos boutons d'option en ajoutant le signe & devant la lettre que vous souhaitez utiliser comme touche d'accès rapide.
Ex : If Option1(0).Value = True Then Label5.Caption = "+"

Afficher une aide simple

Vous pouvez créer une autre feuille et y créer des boîtes **Label** et un bouton (**CommandButton**) pour sortir.

Mais si vous avez plusieurs lignes de commentaires, cela est un peu tristounet.

Vous avez la possibilité de créer un fichier **.RTF** avec les avantages qu'il procure (police, couleurs ...)

Créer un fichier .HLP (voir Help WorkShop) puis, au début de votre feuille sous **Option Explicit** créez la ligne suivante :

```
Private Declare Function WinHelp% Lib "user32" Alias "WinHelpA" _
 (ByVal hwnd&, ByVal HelpFile$, ByVal wCommand%, dwData As Any)
```

Et dans le **Sub** concerné ce qui suit :

```
Dim nRet
If Len(App.HelpFile) = 0 Then
  MsgBox "Impossible d'affichiez le sommaire de l'aide. Il n 'y a pas
d'aide associée à ce projet.", _
    vbInformation, Me.Caption
Else
  On Error Resume Next
  nRet = WinHelp(Me.hwnd, App.HelpFile, 3, 0)
  If Err Then
    MsgBox Err.Description
  End If
End If
```

Dans l'**Explorateur de projets** (Menu/Affichage/Explorateur de projets, s'il n'est pas affiché), clic droit le le projet en court, **Propriété**, mettez le chemin+nom de votre fichier **.HLP**.

Ne me demandez pas plus d'explications, je n'ai pas très bien compris, mais çà marche.

DEBUT

Les tests

`If ... Then ...`	`If ... Then`
	`  ...`
	`  ElseIf ... Then`
	`Else`
	`  ...`
	`End If`
		`Select Case Nombre  ' Évalue Nombre.`
		`  Case 1 To 5`
		`  MsgBox "Le nombre est compris entre 1 et 5 inclus"`
		`  Case 6, 7, 8`
		`  MsgBox "Le nombre est compris entre 6 et 8 inclus"`
		`  Case 9 To 10`
		`  MsgBox " Le nombre est 9 ou 10."`
		`  Case Else`
		`  MsgBox "Non compris entre 1 et 10"`
		`End Select`
			`Select Case Button.Key`
			`  Case "Sauve"`
			`  Case "Couper"`
			`  Case "Coller"`
			`  Case "Ncr"`
			`End Select`

DEBUT

Les Contrôles de base de la boite à outils

Nom du contrôle	Description
 PictureBox	C'est un conteneur d' autres objets. Vous pouvez insérer une image ou par exemple un groupe d' objets en une seule opération

Label	On utilise ce contrôle pour placer du texte comme affichage simple ou comme étiquette.(L' utilisateur ne peut pas modifier le texte saisi dans ce contrôle)
TextBox	C' est le seul objet qui vous permet de saisir du texte, des nombres ou des dates.
Frame	Ce contrôle sert à enjoliver votre présentation. Un autre contrôle remplit le même rôle mais n' est pas livré en standard, c 'est le contrôle Sheridan 3D Controls qui est généré par le fichier Threed32.ocx.(A posséder impérativement)
CommandButton	C'est un bouton poussoir qui enclenchera une action par l' intermédiaire d' une procédure événementielle.
CheckBox	C'est une case à cocher
OptionButton	C'est un bouton radio(Option)
ComboBox	Il s' agit d' une liste modifiable qui permet de choisir un seul élément dans une liste mais aussi de taper une valeur qui n' est pas affichée
ListBox	C'est une liste dans laquelle vous pouvez choisir un ou plusieurs éléments, sans pouvoir saisir de nouvelle valeur.
HScrollBar	C'est un ascenseur horizontal
VScrollBar	C'est un ascenseur vertical
Timer	C'est une minuterie, genre chronomètre qui vous permet d' enclencher une action toutes les n millisecondes
DriveListBox	Ce contrôle permet d' afficher la liste de tous les lecteurs disponibles sur l' ordinateur
DirListBox	Ce contrôle permet d' afficher la liste de tous les répertoires d' un lecteur sélectionné
FileListBox	Ce contrôle permet d' afficher la liste de tous les fichiers d' un répertoire sélectionné
Shape	Permet de dessiner des formes, rectangles, cercles
Line	Permet de dessiner des lignes
Image	Sert à insérer des images, nous le préférerons à l' objet PictureBox
Data	Ce contrôle permet de programmer l' accès aux bases de données
OLE	Il permet de placer des applications OLE(Object Link and Embedding)

Les boites de dialogues communes

Les boites de dialogues communes Il existe 5 boîtes de dialogue communes en Visual Basic :

- d'ouverture
- d'enregistrement
- de couleur
- de police de caractères
- d'impression

Pour cela il faut d'abord ajouter le contrôle CommonDialog dans votre boîte à outils (clic droit sur boite à outil, composants, cochez la case du contrôle Microsoft Common Dialog Control 6.0 (SP3) puis Appliquer).

1. La boîte de dialogue Ouvrir

Cet la boîte de dialogue "Ouvrir" standard de Windows. On fait appel à la méthode ShowOpen. Cependant, il est aussi nécessaire de renseigner plusieurs propriétés de la boîte de dialogue :

Propriété	Utilisation
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL) lorsque le bouton Annuler a été pressé.
DefaultExt	définit l'extension par défaut du nom du fichier à ouvrir.
DialogTitle	définit le titre de la boîte de dialogue situé sur la barre de titre.
FileName	définit le chemin d'accès et le nom du fichier sélectionné par défaut.
Filter	définit le(s) filtre(s) qui sert à spécifier quel type de fichier pouvant être ouvert en lecture. Par exemple, avec l'instruction " CMD.Filter =" DLL (*.DLL)|*.DLL|Exécutables (*.EXE)|*.EXE|Tous (*.*)|*.*| ", vous pouvez choisir de n'ouvrir que les fichiers DLL ou Exécutables ou bien d'ouvrir tous les fichiers.
FilterIndex	spécifie le filtre à utiliser par défaut dans la boîte de dialogue. Reprenons l'instruction précédente : avec l'instruction suivante "CMD.FilterIndex = 2", le filtre utilisé par défaut sera donc Exécutables (*.EXE)|*.EXE|.
Flags	définit les options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& + &H1&".
InitDir	définit le répertoire des fichiers affichés à l'ouverture de la boîte de dialogue.

2. La boîte de dialogue Sauvegarder sous.

La boîte de dialogue "Enregistrer sous" permet de sauvegarder un fichier ouvert. Pour l'afficher, on fait appel à la méthode ShowSave. Pour ce qui est des propriétés à modifier, reportez-vous à ceux de la boîte de dialogue "Ouvrir".

3. La boîte de dialogue Couleur.

Cette boîte de dialogue "Couleur" standard permet une couleur parmi d'autres. Pour l'afficher, on fait appel à la méthode ShowColor. Vous devrez renseigner les propriétés du contrôle suivantes.

Propriété	Utilisation
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL) lorsque le bouton Annuler a été pressé.
Flags	Elle définit les options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& + &H1&".
Color	Elle sélectionne une couleur par défaut. La syntaxe de l'instruction est: "Objet.Color = QBColor(valeur)".

4. La boîte de dialogue Police de caractères

Boîte de dialogue "Police de caractères" standard, permet de sélectionner une police et ses attributs. Pour l'afficher, on fait appel à la méthode ShowFont. Ensuite, renseignez les propriétés suivantes:.

Propriété	Utilisation
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL) lorsque le bouton Annuler a été pressé
Flags	Options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& + &H1&".
FontBold	style Gras par défaut.
FontItalic	style Italique par défaut.
FontName	police par défaut.
FontSize	taille par défaut.
FontStrikethru	style Barré par défaut.
FontUnderline	style Souligné par défaut.
Max	taille maximale des polices affichés.
Min	taille minimale des polices affichés.

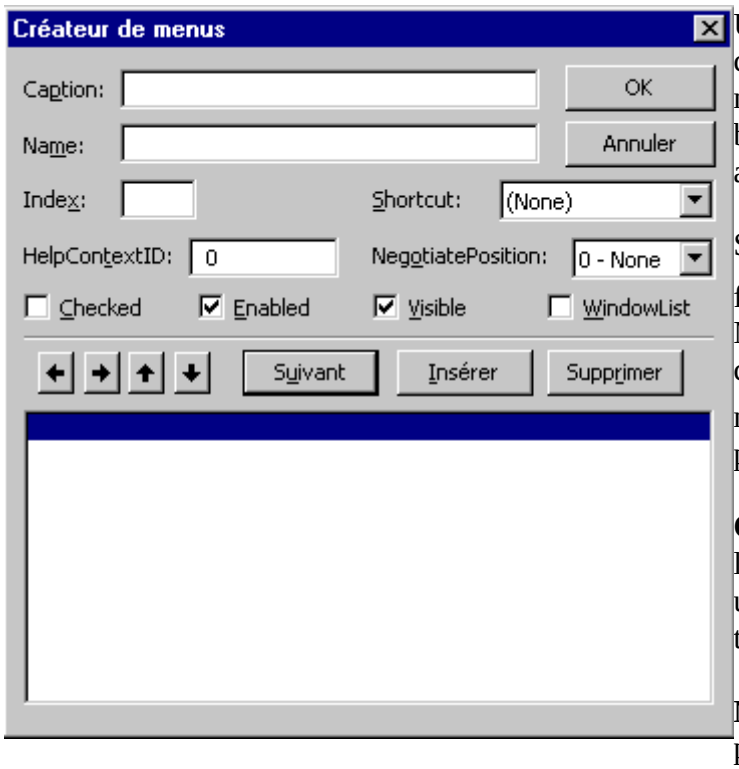
2. La boîte de dialogue Imprimer

La boîte de dialogue "Imprimer" sert à définir les paramètres de l'impression. Pour l'afficher, on fait appel à la méthode ShowPrinter. Ensuite, renseignez les propriétés suivantes:

Propriété	Utilisation
CancelError	Initialisée à True, elle renvoie le code d'erreur 32755 (CDERR_CANCEL)

	lorsque le bouton Annuler a été pressé.
Copies	Nombre d'exemplaires à imprimer.
Flags	Options de la boîte de dialogue. La syntaxe de l'instruction est : "Objet.Flags = valeur" où "valeur" peut valoir "&H1&", "&H2&", "&H3&" ou bien "&H2& + &H1&".
FromPage	Numéro de la première page à imprimer.
PrinterDefault	Indiquer si les paramètres entrés doivent devenir les nouveaux paramètres par défaut.
ToPage	Numéro de la dernière page à imprimer.

Menus et menus contextuels



Un menu peut inclure des commandes, des titres de sous-menus (jusqu'à quatre niveaux)et des barres de séparation. Il doit être associé à une feuille (Form)

Sélectionnez une feuille dans la fenêtre projet, puis son objet . Maintenant vous pouvez accéder à la création de menu (Outils/Créateur de menus) ou par l'icône (3ème à partir de la droite).

Caption : Nom qui apparaîtra à l'utilisateur. Le trait d'union (-) crée une barre séparatrice, & pour la touche Alt.

Name : nom identificateur pour le programme. Il n'apparaît pas à l'utilisateur.

Index : pour affecter une valeur numérique qui détermine la position du contrôle à l'intérieur d'un groupe de contrôles. Cette position n'a aucun rapport avec la position à l'écran.

ShortCut : raccourci clavier pour chaque commande.

HelpContextID : Vous permet d'affecter une valeur numérique unique pour l'identificateur de contexte. Cette valeur est utilisée pour trouver la rubrique appropriée dans le fichier d'aide identifié par la propriété HelpFile.

NegotiatePosition : Vous permet de sélectionner la propriété NegotiatePosition du menu. Cette propriété détermine si le menu apparaît dans une feuille conteneur, et, si oui, à quel emplacement.

Checked : Vous permet d'ajouter une coche à la gauche d'un élément de menu. Celle-ci est généralement utilisée pour signaler si une option à bascule est validée ou non.

Enabled : Vous permet de décider si l'élément de menu doit répondre à des événements, ou doit être grisé si vous voulez qu'il soit indisponible.

Visible : Vous permet de rendre l'élément visible dans le menu.

WindowList : Détermine si le contrôle Menu contient une liste des feuilles MDI fille ouvertes dans une application MDI (càd afficher les derniers fichiers ouverts par l'application concernée).

Le menu contextuel

Dans une quelconque application de Microsoft, lorsque vous cliquez sur le bouton droit de votre souris, une série de commande apparaît: c'est le menu contextuel (ou Popup menu en anglais).
La création d'un popup menu se fait presque comme un système de menu. Pour cela, faites exactement comme si vous créez un système de menu. Définissez ensuite, une action pour chaque commande du menu contextuel. Par la suite, il faut définir dans quelles conditions doit apparaître le menu contextuel. Double-cliquez sur la feuille où est créé le système de menu et dans la liste déroulante "événement", sélectionnez l'événement "MouseDown". Entrez les instructions suivantes :

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As
Single), Y As Single
If (Button = 2) Then
PopupMenu X
end if End Sub
```

Interprétation : Lorsque le bouton droit de la souris a été cliqué, le menu contextuel X apparaît.
Vous pouvez remplacer la valeur de la variable "Button" si vous voulez que le menu contextuel apparaisse à une autre action de l'utilisateur :

- Pour que le menu contextuel apparaisse en cliquant sur le bouton gauche de la souris, remplacez 2 par 1 dans la condition (Button = 1);
- Pour que le menu contextuel apparaisse en cliquant sur le bouton central de la souris, remplacez 2 par 4 dans la condition (Button = 4);

Enfin, revenez dans la boîte de dialogue "créateur de menu" et décochez la case "Visible" du menu que vous désirez rendre contextuel. Ne le faites pas avant car si vous le faites, vous ne pourrez pas définir une action pour chaque commande du menu contextuel.

```
Private Sub RText1_MouseUp(Button As Integer, Shift As Integer, X As
Single, Y As Single)
    If Button = vbRightButton Then
        PopupMenu mnuEdit
    End If
End Sub
```

DEBUT

Le presse papier

Le presse-papier est indispensable pour toute opération de copie et de collage. Ces opérations s'effectuent grâce à l'objet ClipBoard.

Remarque :

- Si vous utilisez le contrôle TextBox, vous n'avez pas besoin de définir des procédures pour le presse-papier (il est déjà intégré dans le contrôle).
- Si vous utilisez le contrôle RichtextBox, alors il vous faudra définir vous-même le presse-papier.

La différence entre ces 2 contrôles de saisie est que le contrôle RichTextBox possède quelques fonctions supplémentaires comme la possibilité de définir des marges autour du texte et surtout il peut supporter des fichiers Txt > 60Ko.

Méthodes utilisées pour définir le presse-papier :

- ClipBoard.GetText : lire le contenu du presse-papier(uniquement avec des données textuelles).
- ClipBoard.SetText : écrire dans le presse-papier(uniquement avec des données textuelles).
- ClipBoard.Clear : effacer le contenu du presse-papier.
- ClipBoard.GetFormat(type) : indique le type de données présent dans le presse-papier. Les valeurs retournée sont :
    - (vbCFText) : données de type texte.
    - (vbCFBitmap) : image en mode point(bitmap).
    - (vbCFMetafile) : métafichier vectoriel(wmf).
    - vbCFDib) : image bitmap indépendante du périphérique.
    - (vbCFPalette) : palette de couleur.
    - (vbCFEMetafile) : métafichier amélioré(emf).
    - (vbCFFile) : noms de fichiers copiés depuis l'Explorateur de Windows.
    - &FFFFBF00 (vbCFLink) : liaisonDDE(valeur exprimée sous forme hexadécimale).
    - &FFFFBF01 (vbCFRTF) : texte au format RTF.
    - ClipBoard.GetData permet de lire le contenu du presse-papier(uniquement avec des données graphiques).
    - ClipBoard.SetData permet d'écrire dans le presse-papier(uniquement avec des données graphiques).

La barre d'outils et la barre d'état

Il faut que soit présent le contrôle Microsoft Windows Common Controls 6.0.

1. La barre d'outils

Créez en une, elle se placera automatiquement sur le haut de votre feuille.
Sélectionnez le contrôle ImageList et placez-le à son tour sur la feuille. (Notez que l'emplacement de ce contrôle n'a aucune importance.)

Clic droit sur le contrôle ImageList sur la feuille, puis Propriétés/onglet "Général". Vous avez la possibilité de modifier la taille de vos images présentes dans la barre d'outils. C'est dans l'onglet "Images" que vous allez pouvoir choisir les images qui y seront placées. Appuyez sur le bouton "Insérer une image" (celles fournies avec Visual Basic sont situés dans C:\Program Files\Microsoft Visual Studio\Common\Graphics\Bitmaps), sélectionnez les images que désirez mettre sur votre barre d'outils. Attribuez à chacune de ces images un nom dans la propriété "Key". Validez ensuite en appuyant sur le bouton "Appliquer".

Clic droit sur le contrôle ToolBar et sélectionnez "propriétés". Dans l'onglet "Général", sélectionnez "ImageList1" dans les propriétés "ImageList", "DisabledImageList" et "HotImageList". sélectionnez aussi "1-TbrFlat" dans la propriété "Style".
Passons à présent à l'onglet "Boutons". Renseignez les principales propriétés :

- Key : le nom des images que vous avez choisi dans le contrôle ImagaList.
- ToolTipText : vous permet d'afficher une bulle lorsque vous survolez l'image.
- Image : les numéros correspondants aux images contenues dans le contrôle ImagaList.

N'oubliez pas que ce dernier permet d'afficher des images sur votre barre d'outils. Ok, maintenant, revenez à la feuille principale. Vous avez vu, votre barre d'outils est à présent visible.

2. La barre d'état

Choisissez le contrôle StatusBar et placez-le n'importe où. cela n'a pas d'importance car au bout du compte, il s'affichera automatiquement tout en bas de votre feuille. Faites un clic droit sur ce dernier qui est à présent placé sur la feuille et allez dans l'onglet "Propriétés". Paramétrez les différentes propriétés qui s'y trouvent.

Les imprimantes


```
For I = 0 To Printers.Count - 1
    Txt.Text = Txt.Text & "------ Imprimante " & (I + 1) & " ------" &
vbNewLine
    Txt.Text = Txt.Text & "Device Name =" & Printers(I).DeviceName &
vbNewLine
    Txt.Text = Txt.Text & "Driver Name =" & Printers(I).DriverName &
vbNewLine
    Txt.Text = Txt.Text & "Port =" & Printers(I).Port & vbNewLine
Next
Set Printer = Printer (0)
```

Appel de base sous SQL Server

Fig.1

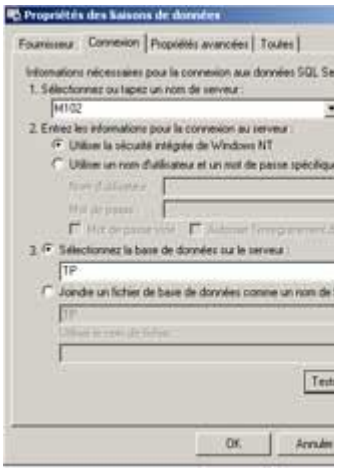
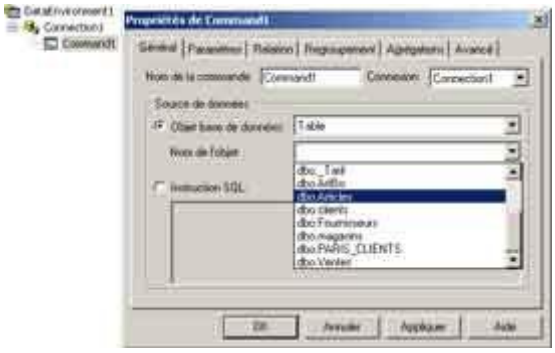


Fig.2

- Si dans la boite à outils l'objet Data (Data Environnement) n'existe pas : clic droit sur la boite à outils, composant, onglet concepteurs, cocher Data Environnement.
- Puis clic droit sur le nom du projet, ajouter, Data Environnement (fig.1)
- Double clic sur Data Environnement
- dans la grande fenêtre clic droit sur connection1, propriétés (fig.2) :
    - Onglet Fournisseur choisir Microsoft OLE DB Provider For SQL Server (pour SQL sever 2000)
    - Onglet connection
        - choisir sa connection (ex M102) dans la liste déroulante
        - sélectionner Utiliser la sécurité intégrée de Windows NT (option Bouton)
        - sélectionner sélectionner la base de donnée sur le serveur (option Bouton)
        - Choisir la base sur laquelle vous voulez travailler dans la liste déroulante
        - Cliquer sur le bouton tester la connection pour vérifier
        - Cliquer sur le bouton "OK"

Dans la fenêtre principale (celle du data environnement, si besoin aller dans le menu principal, fenêtre) :



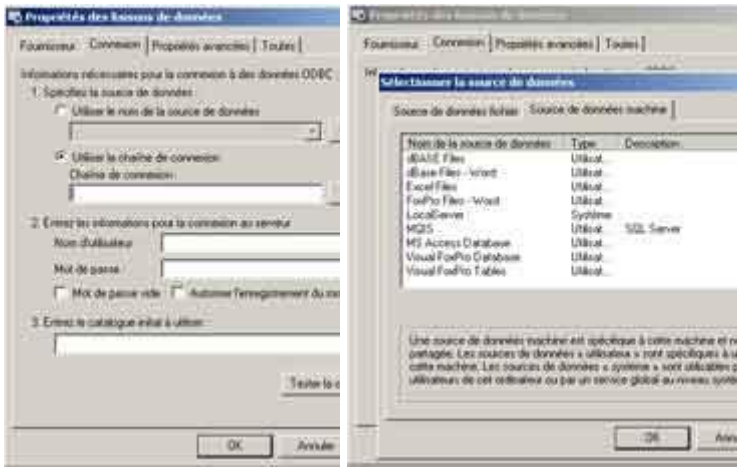
- clic droit sur la connection et choisir ajouter une commande.
- clic droit sur ce dernier, puis propriété là, vous pouvez
  - soit choisir directement l'Objet de base de donnée (table par ex.) et le nom de l'objet dans listes déroulantes (fig.4)
  - soit choisir instructions SQL, cliquez sur le bouton Générateur SQL pour créer votre propre vue.

Votre table est prête à l'emploie. Il vous ne reste plus qu'à transporter les champs sur la feuille (form) VB.

- Mettez côte à côte les fenêtres Form et DataEnvironnement
- Faites un cliquer-glisser de la commande (ou du détail, +) de la feuille DataEnvironnement à la Form.

Note : pour vous aider, dans le menu, affichage, sélectionner "Fenêtre Données". Dans cette dernière dérouler la connection (+) pour avoir accès aux tables.

Appel d'une base Access



- Si dans la boite à outils l'objet Data (Data Environnement) n'existe pas : clic droit sur la boite à outils, composant, onglet concepteurs, cocher Data Environnement.
- Puis clic droit sur le nom du projet, ajouter, Data Environnement (idem SQL fig.1)
- Double clic sur Data Environnement
- dans la grande fenêtre clic droit sur connection1, propriétés :

    - Onglet Fournisseur choisir Microsoft OLE DB Provider for ODBC Drivers
    - Onglet connection
        - choisir utiliser la chaîne de connection
        - Cliquer sur le bouton
        - Onglet Source de données machine : Choisir MS Access Database
        - Sous la nouvelle fenêtre cliquer sur Base de donnée, et choisir sa base Access, valider
        - Cliquer sur le bouton tester la connection pour vérifier
        - Cliquer sur le bouton "OK"

Dans la fenêtre principale (celle du data environnement, si besoin aller dans le menu principal, fenêtre) :

- clic droit sur la connection et choisir ajouter une commande.
- clic droit sur ce dernier, puis propriété, puis :
    - Cliquez sur l'onglet relation (un message apparaîtra, n'en tenez pas compte), choisir votre base de donnée Access.
    - Ne sortez pas de la fenêtre, cliquez sur l'onglet général et choississez votre objet base de donnée (ex : table) ou créez-en un Instruction SQL.
    - Cliquez sur Appliquer, puis sur OK.
    - Faire un cliquer-glisser de la commande ou des champs sur la Form (idem que pour SQL Server)

ADO Data Control : l'objet

Dans la boîte à outils choisir le composant Microsoft ADO Data control (différent suivant version ex : 6.0 SP4). Cliquer sur l'objet  et placez-le sur votre feuille Form.

Clic droit sur cet objet, propriété :

- Onglet général : cliquez sur le bouton céer (une chaîne de connexion) choissisez le fournisseur et la connection (voir accès au base SQL ou accès aux bases Access).

- Onglet ressource, fenêtre type de commande:
  - Soit AdCmdUnknown pour créer votre propre requête
  - Soit AdCmdTable pour choisir la Table
  - soit ....

Pour afficher les champs, créez une TexBox, puis dans ses propriétés :

- DataSource choisir dans la liste votre AdOdc
- DataField choisir dans la liste un champ

ADO Data Control : l'objet

- Connection (source de données)
- Command (permet des requêtes et envoyer des enregistrements)
- Recordset (gère les enregistrements, Fields = champs)

Provider :

- SQLOLEDB pour SQL Server
- MSDASQL pour 1 source ODBC
- Microsoft.jet.OLEDB.3.51 (ou 4.0) pour le moteur Jet
- MSIDXS pour Index Server
- ADSDSOObjet pour Active Directory Service
- MSDAORA pour Oracle

Ex :

```
Dim Cnx As ADODB.Connection
Set Cnx = New Connection
With Cnx
  .Provider = "SQLOLEDB"
  .ConnectionString = "User ID=toto;Data Source = SQL_SCR_1;Initial
Catalog=Mabase"
  .open
End With
Dim Cmd As ADODB.Command
Set Cmd = New Command
With Cmd
  .ActiveConnection = Cnx
  .CommandText = "Update Vente SET px = px * 0.1"
  .Execute
End Wit
```

ou

```
Dim Rs As New Recordset
...
```

```
Private Sub Form_Load()
  Rs.Open "SELECT * FROM " & nom_table, ma_connexion, adOpenDynamic,
adLockOptimistic
```

ou bien sans définir de connection

```
Dim Cmd As ADODB.Command
Set Cmd = New Command
With Cmd
  .ActiveConnection = "Provider = "SQLOLEDB;User ID=toto;Data Source =
SQL_SCR_1;Initial Catalog=Mabase"
  ...
End With
```

SetFocus : bug

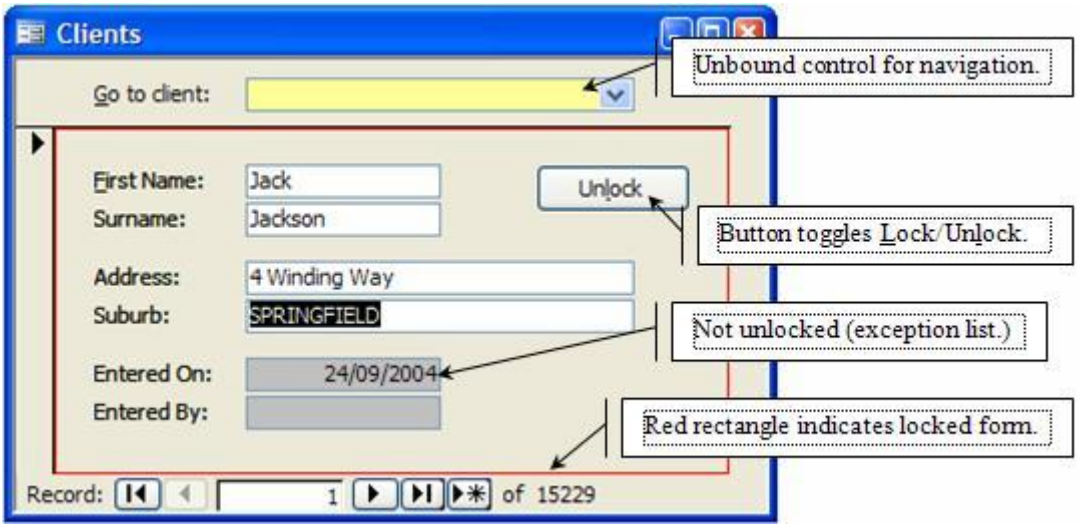
On ne peut pas modifier le focus d'une zone dans la form.load, utilisez Tableindex = 0

Appel prog. ext. : Shell

Help WorkShop

Locking bound controls

It is very easy to overwrite data accidentally in Access. Setting a form's *AllowEdits* property prevents that, but also locks any unbound controls you want to use for filtering or navigation. This solution locks only the bound controls on a form and handles its subforms as well.



First, the code saves any edits in progress, so the user is not stuck with a half-edited form. Next it loops through all controls on the form, setting the *Locked* property of each one **unless** the control:

- is an unsuitable type (lines, labels, ...);
- has no *Control Source* property (buttons in an option group);

- is bound to an expression (*Control Source* starts with "=");
- is unbound (*Control Source* is blank);
- is named in the exception list. (You can specify controls you do not want unlocked.)

If it finds a **subform**, the function calls itself **recursively**. Nested subforms are therefore handled to any depth. If you do not want your subform locked, name it in the exception list.

The form's *AllowDeletions* property is toggled as well. The code changes the text on the command button to indicate whether clicking again will lock or unlock.

To help the user remember they must unlock the form to edit, add a rectangle named rctLock around the edge of your form. The code shows this rectangle when the form is locked, and hides it when unlocked.

Using with your forms

To use the code:

1. Open a new module. In Access 95 - 2003, click the Modules tab of the Database window, and click New. In Access 2007 and later, click Module (rightmost icon) on the Create ribbon. Access opens a code module.
2. Paste in the code from the end of this article.
   Save the module with a name such as ajbLockBound.
3. (Optional) Add a red rectangle to your form to indicate it is locked. Name it rctLock.
4. To initialize the form so it comes up locked, set the *On Load* property of your form to:
   `=LockBoundControls([Form],True)`
5. Add a command button to your form. Name it cmdLock.
   Set its *On Click* property to `[Event Procedure]`.
   Click the Build button (...) beside this.
   Set up the code like this:

```
6.      Private Sub cmdLock_Click()
7.          Dim bLock As Boolean
8.          bLock = IIf(Me.cmdLock.Caption = "&Lock", True, False)
9.          Call LockBoundControls(Me, bLock)
        End Sub
```

10. (Optional) Add the names of any controls you do not want unlocked at steps 3 and 4. For example, to avoid unlocking controls EnteredOn and EnteredBy in the screenshot above, you would use:

```
         Call LockBoundControls(Me, bLock, "EnteredOn",
"EnteredBy")
```

Note that if your form has any disabled controls, changing their *Locked* property affects the way they look. To avoid this, add them to the exception list.

The code

```
Public Function LockBoundControls(frm As Form, bLock As Boolean, ParamArray
avarExceptionList())
On Error GoTo Err_Handler
    'Purpose:   Lock the bound controls and prevent deletes on the form any
its subforms.
```

```
    'Arguments  frm = the form to be locked
    '           bLock = True to lock, False to unlock.
    '           avarExceptionList: Names of the controls NOT to lock
(variant array of strings).
    'Usage:     Call LockBoundControls(Me. True)
    Dim ctl As Control      'Each control on the form
    Dim lngI As Long        'Loop controller.
    Dim bSkip As Boolean

    'Save any edits.
    If frm.Dirty Then
        frm.Dirty = False
    End If
    'Block deletions.
    frm.AllowDeletions = Not bLock

    For Each ctl In frm.Controls
        Select Case ctl.ControlType
        Case acTextBox, acComboBox, acListBox, acOptionGroup, acCheckBox,
acOptionButton, acToggleButton
            'Lock/unlock these controls if bound to fields.
            bSkip = False
            For lngI = LBound(avarExceptionList) To
UBound(avarExceptionList)
                If avarExceptionList(lngI) = ctl.Name Then
                    bSkip = True
                    Exit For
                End If
            Next
            If Not bSkip Then
                If HasProperty(ctl, "ControlSource") Then
                    If Len(ctl.ControlSource) > 0 And Not ctl.ControlSource
Like "=*" Then
                        If ctl.Locked <> bLock Then
                            ctl.Locked = bLock
                        End If
                    End If
                End If
            End If

        Case acSubform
            'Recursive call to handle all subforms.
            bSkip = False
            For lngI = LBound(avarExceptionList) To
UBound(avarExceptionList)
                If avarExceptionList(lngI) = ctl.Name Then
                    bSkip = True
                    Exit For
                End If
            Next
            If Not bSkip Then
                If Len(Nz(ctl.SourceObject, vbNullString)) > 0 Then
                    ctl.Form.AllowDeletions = Not bLock
                    ctl.Form.AllowAdditions = Not bLock
                    Call LockBoundControls(ctl.Form, bLock)
                End If
            End If

        Case acLabel, acLine, acRectangle, acCommandButton, acTabCtl,
acPage, acPageBreak, acImage, acObjectFrame
            'Do nothing
```

```
        Case Else
            'Includes acBoundObjectFrame, acCustomControl
            Debug.Print ctl.Name & " not handled " & Now()
        End Select
    Next

    'Set the visual indicators on the form.
    On Error Resume Next
    frm.cmdLock.Caption = IIf(bLock, "Un&lock", "&Lock")
    frm!rctLock.Visible = bLock


Exit_Handler:
    Set ctl = Nothing
    Exit Function

Err_Handler:
    MsgBox "Error " & Err.Number & " - " & Err.Description
    Resume Exit_Handler
End Function

Public Function HasProperty(obj As Object, strPropName As String) As
Boolean
    'Purpose:   Return true if the object has the property.
    Dim varDummy As Variant
    On Error Resume Next
    varDummy = obj.Properties(strPropName)
    HasProperty = (Err.Number = 0)
End Function
```

Using a Combo Box to Find Records

Warning: In Access 97 or earlier, this tip can trigger the Bookmark Bug.
If you receive an error in Access 2000 or later, see Solving problems with References.

It is possible to use an **unbound combo** box in the header of a form as a means of **record navigation**. The idea is to select an entry from the drop-down list, and have Access take you to that record.

Assume you have a table called "tblCustomers" with the following structure:

```
CustomerID       AutoNumber (indexed as Primary Key).
Company          Text
ContactPerson    Text
```

A form displays data from this table in Single Form view. Add a combo box to the form's header, with the following properties:

```
Name             cboMoveTo
Control Source   [leave this blank]
Row Source Type  Table/Query
Row Source       tblCustomers
Column Count     3
Column Widths    0.6 in; 1.2 in; 1.2 in
```

```
   Bound Column      1
   List Width        3.2 in
   Limit to List     Yes
```

Now attach this code to the AfterUpdate property of the Combo Box:

```
Sub CboMoveTo_AfterUpdate ()
    Dim rs As DAO.Recordset

    If Not IsNull(Me.cboMoveTo) Then
        'Save before move.
        If Me.Dirty Then
            Me.Dirty = False
        End If
        'Search in the clone set.
        Set rs = Me.RecordsetClone
        rs.FindFirst "[CustomerID] = " & Me.cboMoveTo
        If rs.NoMatch Then
            MsgBox "Not found: filtered?"
        Else
            'Display the found record in the form.
            Me.Bookmark = rs.Bookmark
        End If
        Set rs = Nothing
    End If
End Sub
```

The steps this procedure takes are:

- **Save** the record. (This prevents a bug if the record is dirty and can't be saved, e.g. a required field is missing.)
- Refer to **RecordsetClone** - a duplicate set of pointers to the records behind this form ("Me").
- Use **FindFirst** to match the field to be searched with the contents of the combo box.
- **Display** the record, by matching the form's **BookMark** to the record found in the clone recordset.

Note: If CustomerID is a *Text* type field in your table, you need extra quotes, i.e.:

```
    rs.FindFirst "[CustomerID] = """ & Me.cboMoveTo & """"
```

For an explanation, see Quotation marks within quotes.

If you want to create a search form, not merely a navigation combo, see the Search Criteria database.

Archive: Move Records to Another Table

A move consists of two action queries: **Append and Delete**. A **transaction** blocks the Delete if the Append did not succeed. Transactions are not difficult, but there are several pitfalls.

Should I archive?

Probably not. If possible, keep the old records in the same table with the current ones, and use a field to distinguish their status. This makes it much easier to query the data, compare current with old values, etc. It's possible to get the data from different tables back together again with

UNION statements, but it's slower, can't be displayed as a graphic query, and the results are read-only.

Archiving is best reserved for cases where you **won't ever need the old data**, or there are overriding considerations e.g. **hundreds of thousands of records, with new ones being added constantly.** The archive table will probably be in a separate database.

The Steps

The procedure below consists of these steps:

1. Start a **transaction**.
2. Execute the **append** query.
3. Execute the **delete** query.
4. Get user confirmation to **commit** the change.
5. If anything went wrong at any step, **roll back** the transaction.

The Traps

Watch out for these serious traps when working with transactions:

1. **Use dbFailOnError** with the Execute method. Otherwise you are not notified of any errors, and the results could be incomplete.
2. **dbFailOnError without a transaction is not enough.** In Access 95 and earlier, dbFailOnError rolled the entire operation back, and the Access 97 help file wrongly claims that is still the case. (There is a correction in the readme.) FromAccess 97 onwards, dbFailOnError stops further processing when an error occurs, but everything up to the point where the error occurred is committed.
3. **Don't close the default workspace!** The default workspace--dbEngine(0)--is always open. You will set a reference to it, but you are not opening it. Access will allow you to close it, but later you will receive unrelated errors about objects that are no longer set or have gone out of scope. Remember: *Close only what you open; set all objects to nothing.*
4. **CommitTrans or Rollback**, even after an error. The default workspace is always open, so an unterminated transaction remains active even after your procedure ends! And since Access supports multiple transactions, you can dig yourself in further and further. Error handling is essential, with the rollback in the error recovery section. A flag indicating whether you have a transaction open is a practical way to manage this.

The Code

This example selects the records from *MyTable* where the field *MyYesNoField* is Yes, and moves them into a table named *MyArchiveTable* in a different database file - *C:\My Documents\MyArchive.mdb*.

Note: Requires a reference to the DAO library.

```
Sub DoArchive()
On Error GoTo Err_DoArchive
```

```
  Dim ws As DAO.Workspace   'Current workspace (for transaction).
  Dim db As DAO.Database    'Inside the transaction.
  Dim bInTrans As Boolean   'Flag that transaction is active.
  Dim strSql As String      'Action query statements.
  Dim strMsg As String      'MsgBox message.

  'Step 1: Initialize database object inside a transaction.
  Set ws = DBEngine(0)
  ws.BeginTrans
  bInTrans = True
  Set db = ws(0)

  'Step 2: Execute the append.
  strSql = "INSERT INTO MyArchiveTable ( MyField, AnotherField, Field3 ) "
& _
    "IN ""C:\My Documents\MyArchive.mdb"" " & _
    "SELECT SomeField, Field2, Field3 FROM MyTable WHERE (MyYesNoField =
True);"
  db.Execute strSql, dbFailOnError

  'Step 3: Execute the delete.
  strSql = "DELETE FROM MyTable WHERE (MyYesNoField = True);"
  db.Execute strSql, dbFailOnError

  'Step 4: Get user confirmation to commit the change.
  strMsg = "Archive " & db.RecordsAffected & " record(s)?"
  If MsgBox(strMsg, vbOKCancel + vbQuestion, "Confirm") = vbOK Then
    ws.CommitTrans
    bInTrans = False
  End If

Exit_DoArchive:
  'Step 5: Clean up
  On Error Resume Next
  Set db = Nothing
  If bInTrans Then   'Rollback if the transaction is active.
    ws.Rollback
  End If
  Set ws = Nothing
Exit Sub

Err_DoArchive:
  MsgBox Err.Description, vbExclamation, "Archiving failed: Error " &
Err.number
  Resume Exit_DoArchive
End Sub
```

DAO Programming Code Examples

This page is a reference for developers, demonstrating how to use the DAO library to programmatically create, delete, modify, and list the objects in Access - the tables, fields, indexes, and relations, queries, and databases - and read or set their properties.

DAO (Data Access Objects) is the native library Microsoft designed to expose the object in Access. All versions have this library set by default, except Access 2000 and 2002, so make sure you have the DAO library reference set if you use those versions.

For more information on why you need to use DAO, see Michael Kaplan's blog posting of July 13 2007: What does DAO have that ADO/ADOx/JRO do not?

For an introduction to DAO, see DAO Object Model. If you are more familiar with ADO or DDL, this comparison of field names may help.

There is no explanation beyond in-line comments, and no error handling in most examples.

Index of Functions	Description
CreateTableDAO()	Create two tables using DAO, illustrating the field types.
ModifyTableDAO()	Add and delete fields to existing tables.
DeleteTableDAO()	Drop a table
MakeGuidTable()	Create a table with a GUID field.
CreateIndexesDAO()	Create primary key, foreign key, and unique indexes; single- and multi-field
DeleteIndexDAO()	Delete indexes
CreateRelationDAO()	Create relations between tables.
DeleteRelationDAO()	Delete relations
DeleteQueryDAO()	Delete a query programmatically.
SetPropertyDAO()	Set a property for an object, creating if necessary.
HasProperty()	Return true if the object has the property.
StandardProperties()	Properties you always want set by default.
ConvertMixedCase()	Convert mixed case name into a name with spaces.
SetFieldDescription()	Assign a Description to a field.
IndexOnField()	Indicate if there is a single-field index.
CreateQueryDAO()	Create a query programmatically.

CreateDatabaseDAO()	Create a new database programmatically, and set its key properties.
ShowDatabaseProps()	List the properties of the current database.
ShowFields()	How to read the fields of a table.
ShowFieldsRS()	How to read the field names and types from a table or query.
FieldTypeName()	Converts the numeric results of DAO fieldtype to text.
DAORecordsetExample()	How to open a recordset and loop through the records.
ShowFormProperties()	Loop through the controls on a form, showing names and properties.
ExecuteInTransaction()	Execute the SQL statement on the current database in a transaction.
GetAutoNumDAO()	Get the name of the AutoNumber field, using DAO.

```
Option Compare Database
Option Explicit

'Constants for examining how a field is indexed.
Private Const intcIndexNone As Integer = 0
Private Const intcIndexGeneral As Integer = 1
Private Const intcIndexUnique As Integer = 3
Private Const intcIndexPrimary As Integer = 7

Function CreateTableDAO()
    'Purpose:   Create two tables using DAO.
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field

    'Initialize the Contractor table.
    Set db = CurrentDb()
    Set tdf = db.CreateTableDef("tblDaoContractor")

    'Specify the fields.
    With tdf
        'AutoNumber: Long with the attribute set.
        Set fld = .CreateField("ContractorID", dbLong)
        fld.Attributes = dbAutoIncrField + dbFixedField
        .Fields.Append fld

        'Text field: maximum 30 characters, and required.
        Set fld = .CreateField("Surname", dbText, 30)
        fld.Required = True
        .Fields.Append fld

        'Text field: maximum 20 characters.
        .Fields.Append .CreateField("FirstName", dbText, 20)

        'Yes/No field.
        .Fields.Append .CreateField("Inactive", dbBoolean)
```

```
    'Currency field.
    .Fields.Append .CreateField("HourlyFee", dbCurrency)

    'Number field.
    .Fields.Append .CreateField("PenaltyRate", dbDouble)

    'Date/Time field with validation rule.
    Set fld = .CreateField("BirthDate", dbDate)
    fld.ValidationRule = "Is Null Or <=Date()"
    fld.ValidationText = "Birth date cannot be future."
    .Fields.Append fld

    'Memo field.
    .Fields.Append .CreateField("Notes", dbMemo)

    'Hyperlink field: memo with the attribute set.
    Set fld = .CreateField("Web", dbMemo)
    fld.Attributes = dbHyperlinkField + dbVariableField
    .Fields.Append fld
End With

'Save the Contractor table.
db.TableDefs.Append tdf
Set fld = Nothing
Set tdf = Nothing
Debug.Print "tblDaoContractor created."

'Initialize the Booking table
Set tdf = db.CreateTableDef("tblDaoBooking")
With tdf
    'Autonumber
    Set fld = .CreateField("BookingID", dbLong)
    fld.Attributes = dbAutoIncrField + dbFixedField
    .Fields.Append fld

    'BookingDate
    .Fields.Append .CreateField("BookingDate", dbDate)

    'ContractorID
    .Fields.Append .CreateField("ContractorID", dbLong)

    'BookingFee
    .Fields.Append .CreateField("BookingFee", dbCurrency)

    'BookingNote: Required.
    Set fld = .CreateField("BookingNote", dbText, 255)
    fld.Required = True
    .Fields.Append fld
End With

'Save the Booking table.
db.TableDefs.Append tdf
Set fld = Nothing
Set tdf = Nothing
Debug.Print "tblDaoBooking created."

'Clean up
Application.RefreshDatabaseWindow   'Show the changes
Set fld = Nothing
Set tdf = Nothing
Set db = Nothing
```

```
End Function

Function ModifyTableDAO()
    'Purpose:   How to add and delete fields to existing tables.
    'Note:      Requires the table created by CreateTableDAO() above.
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field

    'Initialize
    Set db = CurrentDb()

    Set tdf = db.TableDefs("tblDaoContractor")

    'Add a field to the table.
    tdf.Fields.Append tdf.CreateField("TestField", dbText, 80)
    Debug.Print "Field added."

    'Delete a field from the table.
    tdf.Fields.Delete "TestField"
    Debug.Print "Field deleted."

    'Clean up
    Set fld = Nothing
    Set tdf = Nothing
    Set db = Nothing
End Function

Function DeleteTableDAO()
    DBEngine(0)(0).TableDefs.Delete "DaoTest"
End Function

Function MakeGuidTable()
    'Purpose:   How to create a table with a GUID field.
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
    Dim prp As DAO.Property

    Set db = CurrentDb()
    Set tdf = db.CreateTableDef("Table8")
    With tdf
        Set fld = .CreateField("ID", dbGUID)
        fld.Attributes = dbFixedField
        fld.DefaultValue = "GenGUID()"
        .Fields.Append fld
    End With
    db.TableDefs.Append tdf
End Function

Function CreateIndexesDAO()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim ind As DAO.Index

    'Initialize
    Set db = CurrentDb()
    Set tdf = db.TableDefs("tblDaoContractor")

    '1. Primary key index.
    Set ind = tdf.CreateIndex("PrimaryKey")
```

```
    With ind
        .Fields.Append .CreateField("ContractorID")
        .Unique = False
        .Primary = True
    End With
    tdf.Indexes.Append ind

    '2. Single-field index.
    Set ind = tdf.CreateIndex("Inactive")
    ind.Fields.Append ind.CreateField("Inactive")
    tdf.Indexes.Append ind

    '3. Multi-field index.
    Set ind = tdf.CreateIndex("FullName")
    With ind
        .Fields.Append .CreateField("Surname")
        .Fields.Append .CreateField("FirstName")
    End With
    tdf.Indexes.Append ind

    'Refresh the display of this collection.
    tdf.Indexes.Refresh

    'Clean up
    Set ind = Nothing
    Set tdf = Nothing
    Set db = Nothing
    Debug.Print "tblDaoContractor indexes created."
End Function

Function DeleteIndexDAO()
    DBEngine(0)(0).TableDefs("tblDaoContractor").Indexes.Delete "Inactive"
End Function

Function CreateRelationDAO()
    Dim db As DAO.Database
    Dim rel As DAO.Relation
    Dim fld As DAO.Field

    'Initialize
    Set db = CurrentDb()

    'Create a new relation.
    Set rel = db.CreateRelation("tblDaoContractortblDaoBooking")

    'Define its properties.
    With rel
        'Specify the primary table.
        .Table = "tblDaoContractor"
        'Specify the related table.
        .ForeignTable = "tblDaoBooking"
        'Specify attributes for cascading updates and deletes.
        .Attributes = dbRelationUpdateCascade + dbRelationDeleteCascade

        'Add the fields to the relation.
        'Field name in primary table.
        Set fld = .CreateField("ContractorID")
        'Field name in related table.
        fld.ForeignName = "ContractorID"
        'Append the field.
        .Fields.Append fld
```

```
        'Repeat for other fields if a multi-field relation.
    End With

    'Save the newly defined relation to the Relations collection.
    db.Relations.Append rel

    'Clean up
    Set fld = Nothing
    Set rel = Nothing
    Set db = Nothing
    Debug.Print "Relation created."
End Function

Function DeleteRelationDAO()
    DBEngine(0)(0).Relations.Delete "tblDaoContractortblDaoBooking"
End Function

Function DeleteQueryDAO()
    DBEngine(0)(0).QueryDefs.Delete "qryDaoBooking"
End Function

Function SetPropertyDAO(obj As Object, strPropertyName As String, intType
As Integer, _
    varValue As Variant, Optional strErrMsg As String) As Boolean
On Error GoTo ErrHandler
    'Purpose:   Set a property for an object, creating if necessary.
    'Arguments: obj = the object whose property should be set.
    '           strPropertyName = the name of the property to set.
    '           intType = the type of property (needed for creating)
    '           varValue = the value to set this property to.
    '           strErrMsg = string to append any error message to.

    If HasProperty(obj, strPropertyName) Then
        obj.Properties(strPropertyName) = varValue
    Else
        obj.Properties.Append obj.CreateProperty(strPropertyName, intType,
varValue)
    End If
    SetPropertyDAO = True

ExitHandler:
    Exit Function

ErrHandler:
    strErrMsg = strErrMsg & obj.Name & "." & strPropertyName & " not set to
" & varValue & _
        ". Error " & Err.Number & " - " & Err.Description & vbCrLf
    Resume ExitHandler
End Function

Public Function HasProperty(obj As Object, strPropName As String) As
Boolean
    'Purpose:   Return true if the object has the property.
    Dim varDummy As Variant

    On Error Resume Next
    varDummy = obj.Properties(strPropName)
    HasProperty = (Err.Number = 0)
End Function
```

```
Function StandardProperties(strTableName As String)
    'Purpose:   Properties you always want set by default:
    '           TableDef:        Subdatasheets off.
    '           Numeric fields:  Remove Default Value.
    '           Currency fields: Format as currency.
    '           Yes/No fields:   Display as check box. Default to No.
    '           Text/memo/hyperlink: AllowZeroLength off,
    '                                UnicodeCompression on.
    '           All fields:      Add a caption if mixed case.
    'Argument:  Name of the table.
    'Note:      Requires: SetPropertyDAO()
    Dim db As DAO.Database      'Current database.
    Dim tdf As DAO.TableDef     'Table nominated in argument.
    Dim fld As DAO.Field        'Each field.
    Dim strCaption As String    'Field caption.
    Dim strErrMsg As String     'Responses and error messages.

    'Initalize.
    Set db = CurrentDb()
    Set tdf = db.TableDefs(strTableName)

    'Set the table's SubdatasheetName.
    Call SetPropertyDAO(tdf, "SubdatasheetName", dbText, "[None]", _
        strErrMsg)

    For Each fld In tdf.Fields
        'Handle the defaults for the different field types.
        Select Case fld.Type
        Case dbText, dbMemo 'Includes hyperlinks.
            fld.AllowZeroLength = False
            Call SetPropertyDAO(fld, "UnicodeCompression", dbBoolean, _
                True, strErrMsg)
        Case dbCurrency
            fld.DefaultValue = 0
            Call SetPropertyDAO(fld, "Format", dbText, "Currency", _
                strErrMsg)
        Case dbLong, dbInteger, dbByte, dbDouble, dbSingle, dbDecimal
            fld.DefaultValue = vbNullString
        Case dbBoolean
            Call SetPropertyDAO(fld, "DisplayControl", dbInteger, _
                CInt(acCheckBox))
        End Select

        'Set a caption if needed.
        strCaption = ConvertMixedCase(fld.Name)
        If strCaption <> fld.Name Then
            Call SetPropertyDAO(fld, "Caption", dbText, strCaption)
        End If

        'Set the field's Description.
        Call SetFieldDescription(tdf, fld, , strErrMsg)
    Next

    'Clean up.
    Set fld = Nothing
    Set tdf = Nothing
    Set db = Nothing
    If Len(strErrMsg) > 0 Then
        Debug.Print strErrMsg
    Else
        Debug.Print "Properties set for table " & strTableName
```

```
    End If
End Function

Function ConvertMixedCase(ByVal strIn As String) As String
    'Purpose:   Convert mixed case name into a name with spaces.
    'Argument:  String to convert.
    'Return:    String converted by these rules:
    '           1. One space before an upper case letter.
    '           2. Replace underscores with spaces.
    '           3. No spaces between continuing upper case.
    'Example:   "FirstName" or "First_Name" => "First Name".
    Dim lngStart As Long        'Loop through string.
    Dim strOut As String        'Output string.
    Dim boolWasSpace As Boolean 'Last char. was a space.
    Dim boolWasUpper As Boolean 'Last char. was upper case.

    strIn = Trim$(strIn)        'Remove leading/trailing spaces.
    boolWasUpper = True         'Initialize for no first space.

    For lngStart = 1& To Len(strIn)
        Select Case Asc(Mid(strIn, lngStart, 1&))
        Case vbKeyA To vbKeyZ   'Upper case: insert a space.
            If boolWasSpace Or boolWasUpper Then
                strOut = strOut & Mid(strIn, lngStart, 1&)
            Else
                strOut = strOut & " " & Mid(strIn, lngStart, 1&)
            End If
            boolWasSpace = False
            boolWasUpper = True

        Case 95                 'Underscore: replace with space.
            If Not boolWasSpace Then
                strOut = strOut & " "
            End If
            boolWasSpace = True
            boolWasUpper = False

        Case vbKeySpace         'Space: output and set flag.
            If Not boolWasSpace Then
                strOut = strOut & " "
            End If
            boolWasSpace = True
            boolWasUpper = False

        Case Else               'Any other char: output.
            strOut = strOut & Mid(strIn, lngStart, 1&)
            boolWasSpace = False
            boolWasUpper = False
        End Select
    Next

    ConvertMixedCase = strOut
End Function

Function SetFieldDescription(tdf As DAO.TableDef, fld As DAO.Field, _
Optional ByVal strDescrip As String, Optional strErrMsg As String) _
As Boolean
    'Purpose:   Assign a Description to a field.
    'Arguments: tdf = the TableDef the field belongs to.
    '           fld = the field to document.
    '           strDescrip = The description text you want.
```

```
'                       If blank, uses Caption or Name of field.
'           strErrMsg  = string to append any error messages to.
'Notes:     Description includes field size, validation,
'               whether required or unique.

If (fld.Attributes And dbAutoIncrField) > 0& Then
    strDescrip = strDescrip & " Automatically generated " & _
        "unique identifier for this record."
Else
    'If no description supplied, use the field's Caption or Name.
    If Len(strDescrip) = 0& Then
        If HasProperty(fld, "Caption") Then
            If Len(fld.Properties("Caption")) > 0& Then
                strDescrip = fld.Properties("Caption") & "."
            End If
        End If
        If Len(strDescrip) = 0& Then
            strDescrip = fld.Name & "."
        End If
    End If

    'Size of the field.
    'Ignore Date, Memo, Yes/No, Currency, Decimal, GUID,
    '   Hyperlink, OLE Object.
    Select Case fld.Type
    Case dbByte, dbInteger, dbLong
        strDescrip = strDescrip & " Whole number."
    Case dbSingle, dbDouble
        strDescrip = strDescrip & " Fractional number."
    Case dbText
        strDescrip = strDescrip & " " & fld.Size & "-char max."
    End Select

    'Required and/or Unique?
    'Check for single-field index, and Required property.
    Select Case IndexOnField(tdf, fld)
    Case intcIndexPrimary
        strDescrip = strDescrip & " Required. Unique."
    Case intcIndexUnique
        If fld.Required Then
            strDescrip = strDescrip & " Required. Unique."
        Else
            strDescrip = strDescrip & " Unique."
        End If
    Case Else
        If fld.Required Then
            strDescrip = strDescrip & " Required."
        End If
    End Select

    'Validation?
    If Len(fld.ValidationRule) > 0& Then
        If Len(fld.ValidationText) > 0& Then
            strDescrip = strDescrip & " " & fld.ValidationText
        Else
            strDescrip = strDescrip & " " & fld.ValidationRule
        End If
    End If
End If

If Len(strDescrip) > 0& Then
```

```
        strDescrip = Trim$(Left$(strDescrip, 255&))
        SetFieldDescription = SetPropertyDAO(fld, "Description", _
            dbText, strDescrip, strErrMsg)
    End If
End Function

Private Function IndexOnField(tdf As DAO.TableDef, fld As DAO.Field) _
As Integer
    'Purpose:   Indicate if there is a single-field index _
    '               on this field in this table.
    'Return:    The constant indicating the strongest type.
    Dim ind As DAO.Index
    Dim intReturn As Integer

    intReturn = intcIndexNone

    For Each ind In tdf.Indexes
        If ind.Fields.Count = 1 Then
            If ind.Fields(0).Name = fld.Name Then
                If ind.Primary Then
                    intReturn = (intReturn Or intcIndexPrimary)
                ElseIf ind.Unique Then
                    intReturn = (intReturn Or intcIndexUnique)
                Else
                    intReturn = (intReturn Or intcIndexGeneral)
                End If
            End If
        End If
    Next

    'Clean up
    Set ind = Nothing
    IndexOnField = intReturn
End Function

Function CreateQueryDAO()
    'Purpose:   How to create a query
    'Note:      Requires a table named MyTable.
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef

    Set db = CurrentDb()

    'The next line creates and automatically appends the QueryDef.
    Set qdf = db.CreateQueryDef("qryMyTable")

    'Set the SQL property to a string representing a SQL statement.
    qdf.SQL = "SELECT MyTable.* FROM MyTable;"

    'Do not append: QueryDef is automatically appended!

    Set qdf = Nothing
    Set db = Nothing
    Debug.Print "qryMyTable created."
End Function

Function CreateDatabaseDAO()
    'Purpose:   How to create a new database and set key properties.
    Dim dbNew As DAO.Database
    Dim prp As DAO.Property
    Dim strFile As String
```

```
    'Create the new database.
    strFile = "C:\SampleDAO.mdb"
    Set dbNew = DBEngine(0).CreateDatabase(strFile, dbLangGeneral)

    'Create example properties in new database.
    With dbNew
        Set prp = .CreateProperty("Perform Name AutoCorrect", dbLong, 0)
        .Properties.Append prp
        Set prp = .CreateProperty("Track Name AutoCorrect Info", _
            dbLong, 0)
        .Properties.Append prp
    End With

    'Clean up.
    dbNew.Close
    Set prp = Nothing
    Set dbNew = Nothing
    Debug.Print "Created " & strFile
End Function

Function ShowDatabaseProps()
    'Purpose:   List the properies of the current database.
    Dim db As DAO.Database
    Dim prp As DAO.Property

    Set db = CurrentDb()
    For Each prp In db.Properties
        Debug.Print prp.Name
    Next

    Set db = Nothing
End Function

Function ShowFields(strTable As String)
    'Purpose:   How to read the fields of a table.
    'Usage:     Call ShowFields("Table1")
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field

    Set db = CurrentDb()
    Set tdf = db.TableDefs(strTable)
    For Each fld In tdf.Fields
        Debug.Print fld.Name, FieldTypeName(fld)
    Next

    Set fld = Nothing
    Set tdf = Nothing
    Set db = Nothing
End Function

Function ShowFieldsRS(strTable)
    'Purpose:   How to read the field names and types from a table or
query.
    'Usage:     Call ShowFieldsRS("Table1")
    Dim rs As DAO.Recordset
    Dim fld As DAO.Field
    Dim strSql As String
```

```
    strSql = "SELECT " & strTable & ".* FROM " & strTable & " WHERE
(False);"
    Set rs = DBEngine(0)(0).OpenRecordset(strSql)
    For Each fld In rs.Fields
        Debug.Print fld.Name, FieldTypeName(fld), "from " & fld.SourceTable
& "." & fld.SourceField
    Next
    rs.Close
    Set rs = Nothing
End Function

Public Function FieldTypeName(fld As DAO.Field)
    'Purpose: Converts the numeric results of DAO fieldtype to text.
    'Note:    fld.Type is Integer, but the constants are Long.
    Dim strReturn As String         'Name to return

    Select Case CLng(fld.Type)
        Case dbBoolean: strReturn = "Yes/No"            ' 1
        Case dbByte: strReturn = "Byte"                 ' 2
        Case dbInteger: strReturn = "Integer"           ' 3
        Case dbLong                                     ' 4
            If (fld.Attributes And dbAutoIncrField) = 0& Then
                strReturn = "Long Integer"
            Else
                strReturn = "AutoNumber"
            End If
        Case dbCurrency: strReturn = "Currency"         ' 5
        Case dbSingle: strReturn = "Single"             ' 6
        Case dbDouble: strReturn = "Double"             ' 7
        Case dbDate: strReturn = "Date/Time"            ' 8
        Case dbBinary: strReturn = "Binary"             ' 9 (no interface)
        Case dbText                                     '10
            If (fld.Attributes And dbFixedField) = 0& Then
                strReturn = "Text"
            Else
                strReturn = "Text (fixed width)"
            End If
        Case dbLongBinary: strReturn = "OLE Object"     '11
        Case dbMemo                                     '12
            If (fld.Attributes And dbHyperlinkField) = 0& Then
                strReturn = "Memo"
            Else
                strReturn = "Hyperlink"
            End If
        Case dbGUID: strReturn = "GUID"                 '15

        'Attached tables only: cannot create these in JET.
        Case dbBigInt: strReturn = "Big Integer"        '16
        Case dbVarBinary: strReturn = "VarBinary"       '17
        Case dbChar: strReturn = "Char"                 '18
        Case dbNumeric: strReturn = "Numeric"           '19
        Case dbDecimal: strReturn = "Decimal"           '20
        Case dbFloat: strReturn = "Float"               '21
        Case dbTime: strReturn = "Time"                 '22
        Case dbTimeStamp: strReturn = "Time Stamp"      '23

        'Constants for complex types don't work prior to Access 2007.
        Case 101&: strReturn = "Attachment"         'dbAttachment
        Case 102&: strReturn = "Complex Byte"       'dbComplexByte
        Case 103&: strReturn = "Complex Integer"    'dbComplexInteger
        Case 104&: strReturn = "Complex Long"       'dbComplexLong
```

```
        Case 105&: strReturn = "Complex Single"     'dbComplexSingle
        Case 106&: strReturn = "Complex Double"     'dbComplexDouble
        Case 107&: strReturn = "Complex GUID"       'dbComplexGUID
        Case 108&: strReturn = "Complex Decimal"    'dbComplexDecimal
        Case 109&: strReturn = "Complex Text"       'dbComplexText
        Case Else: strReturn = "Field type " & fld.Type & " unknown"
    End Select

    FieldTypeName = strReturn
End Function

Function DAORecordsetExample()
    'Purpose:   How to open a recordset and loop through the records.
    'Note:      Requires a table named MyTable, with a field named MyField.
    Dim rs As DAO.Recordset
    Dim strSql As String

    strSql = "SELECT MyField FROM MyTable;"
    Set rs = DBEngine(0)(0).OpenRecordset(strSql)

    Do While Not rs.EOF
        Debug.Print rs!MyField
        rs.MoveNext
    Loop

    rs.Close
    Set rs = Nothing
End Function

Function ShowFormProperties(strFormName As String)
On Error GoTo Err_Handler
    'Purpose:   Loop through the controls on a form, showing names and
properties.
    'Usage:     Call ShowFormProperties("Form1")
    Dim frm As Form
    Dim ctl As Control
    Dim prp As Property
    Dim strOut As String

    DoCmd.OpenForm strFormName, acDesign, WindowMode:=acHidden
    Set frm = Forms(strFormName)

    For Each ctl In frm
        For Each prp In ctl.Properties
            strOut = strFormName & "." & ctl.Name & "." & prp.Name & ": "
            strOut = strOut & prp.Type & vbTab
            strOut = strOut & prp.Value
            Debug.Print strOut
        Next
        If ctl.ControlType = acTextBox Then Stop
    Next

    Set frm = Nothing
    DoCmd.Close acForm, strFormName, acSaveNo

Exit_Handler:
    Exit Function

Err_Handler:
    Select Case Err.Number
    Case 2186:
```

```
        strOut = strOut & Err.Description
        Resume Next
    Case Else
        MsgBox "Error " & Err.Number & ": " & Err.Description,
vbExclamation, "ShowFormProperties()"
        Resume Exit_Handler
    End Select
End Function

Public Function ExecuteInTransaction(strSql As String, Optional
strConfirmMessage As String) As Long
On Error GoTo Err_Handler
    'Purpose:   Execute the SQL statement on the current database in a
transaction.
    'Return:    RecordsAffected if zero or above.
    'Arguments: strSql = the SQL statement to be executed.
    '           strConfirmMessage = the message to show the user for
confirmation. Number will be added to front.
    '           No confirmation if ZLS.
    '           -1 on error.
    '           -2 on user-cancel.
    Dim ws As DAO.Workspace
    Dim db As DAO.Database
    Dim bInTrans As Boolean
    Dim bCancel As Boolean
    Dim strMsg As String
    Dim lngReturn As Long
    Const lngcUserCancel = -2&

    Set ws = DBEngine(0)
    ws.BeginTrans
    bInTrans = True
    Set db = ws(0)
    db.Execute strSql, dbFailOnError
    lngReturn = db.RecordsAffected
    If strConfirmMessage <> vbNullString Then
        If MsgBox(lngReturn & " " & Trim$(strConfirmMessage), vbOKCancel +
vbQuestion, "Confirm") <> vbOK Then
            bCancel = True
            lngReturn = lngcUserCancel
        End If
    End If

    'Commmit or rollback.
    If bCancel Then
        ws.Rollback
    Else
        ws.CommitTrans
    End If
    bInTrans = False

Exit_Handler:
    ExecuteInTransaction = lngReturn
    On Error Resume Next
    Set db = Nothing
    If bInTrans Then
        ws.Rollback
    End If
    Set ws = Nothing
    Exit Function
```

```
Err_Handler:
    MsgBox "Error " & Err.Number & ": " & Err.Description, vbExclamation,
"ExecuteInTransaction()"
    lngReturn = -1
    Resume Exit_Handler
End Function

Function GetAutoNumDAO(strTable) As String
    'Purpose:   Get the name of the AutoNumber field, using DAO.
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field

    Set db = CurrentDb()
    Set tdf = db.TableDefs(strTable)

    For Each fld In tdf.Fields
        If (fld.Attributes And dbAutoIncrField) <> 0 Then
            GetAutoNumDAO = fld.Name
            Exit For
        End If
    Next

    Set fld = Nothing
    Set tdf = Nothing
    Set db = Nothing
End Function
```

Functions (VBA)

- MinOfList(), MaxOfList(): Get the min/max of a list of values Access 97 and later
- Soundex(): Fuzzy matches - find names that sound alike Access 95 and later
- Age(): Calculate a person's age, from date of birth data Access 95 and later
- Text2Clipboard(), Clipboard2Text() Copy to and from the Windows Clipboard Access 95 and later
- TableInfo(), FieldTypeName(): List the names, types, descriptions of fields in a table Access 95 and later
- DirListBox(): Fill a list box with the files in a directory All versions
- PlaySound(): Play WAV files in Access events All versions
- ELookup() - an extended replacement for DLookup() Access 95 and later
- ParseWord(): Parses the first, last, or n-th word/item from a field/list Access 2000 and later
- FileExists(), FolderExists(): Determine if a file or folder exists Access 95 and later
- ClearList(), SelectAll(): Select or clear all items in a multi-select list box Access 95 and later
- DeleteAllRelationships(): Delete all relations in a database. Useful for repairs Access 95 and later
- CountLines(): How many lines of code in the current database? Access 2000 and later
- InsertAtCursor() - Insert characters at the cursor position Access 95 and later
- GoHyperlink() - Handle warnings, special characters, and errors opening hyperlinks Access 2000 and later
- AdjustDateForYear() - Intelligent handling of dates at the start of a calendar year. Access 2000 and later

- Keep1Open() - Open a switchboard when other forms/reports close. Access 97 and later

MinOfList() and MaxOfList() functions

Access does not have functions like *Min()* and *Max()* in Excel, for selecting the least/greatest value from a list. That makes sense in a relational database, because you store these values in a related table. So Access provides *DMin()* and *DMax()* for retrieving the smallest/largest value from the column in the related table.

Occasionally, you still need to pick the minimum or maximum value from a list. The functions below do that. They work with **numeric** fields, including **currency** and **dates**. They return Null if there was no numeric value in the list.

Using the functions

To create them:

1. Create a new module. In Access 97 - 2003, click the Modules tab of the database window, and click New. In Access 2007 and later, click the Create ribbon, and choose Module (the rightmost icon on the Other group.) Access opens the code window.
2. Copy the code below, and paste into your code window.
3. Check that Access understands the code, by choosing *Compile* on the *Debug* menu.
4. Save the module with a name such as *Module1*.

Use them like any built-in function.

For example, you could put this **in a text box**:

```
=MinOfList(5, -3, Null, 0, 2)
```

Or you could type this into a fresh column of the Field row **in a query** that has three date fields:

```
MaxOfList([OrderDate], [InvoiceDate], [DueDate])
```

```
Function MinOfList(ParamArray varValues()) As Variant
    Dim i As Integer        'Loop controller.
    Dim varMin As Variant   'Smallest value found so far.

    varMin = Null           'Initialize to null

    For i = LBound(varValues) To UBound(varValues)
        If IsNumeric(varValues(i)) Or IsDate(varValues(i)) Then
            If varMin <= varValues(i) Then
                'do nothing
            Else
                varMin = varValues(i)
            End If
        End If
```

```
    Next

    MinOfList = varMin
End Function

Function MaxOfList(ParamArray varValues()) As Variant
    Dim i As Integer        'Loop controller.
    Dim varMax As Variant   'Largest value found so far.

    varMax = Null           'Initialize to null

    For i = LBound(varValues) To UBound(varValues)
        If IsNumeric(varValues(i)) Or IsDate(varValues(i)) Then
            If varMax >= varValues(i) Then
                'do nothing
            Else
                varMax = varValues(i)
            End If
        End If
    Next

    MaxOfList = varMax
End Function
```

Understanding the functions

The *ParamArray* keyword lets you pass in any number of values. The function receives them as an array. You can then examine each value in the array to find the highest or lowest. The *LBound()* and *UBound()* functions indicate how many values were passed in, and the loop visits each member in the array.

Any nulls in the list are ignored: they do not pass the *IsNumeric()* test.

The return value (*varMin* or *VarMax*) is initialized to Null, so the function returns Null if no values are found. It also means that if no values have been found yet, the line:

```
    If varMin <= varValues(i) Then
```

evaluates to Null, and so the *Else* block executes. Since an *If* statement has three possible outcomes - *True*, *False*, and *Null* - a "do nothing" for one is a convenient way to handle the other two. If that is new, see Common errors with Null.

Note that the functions would yield wrong results if the return value was not initialized to Null. VBA initializes it to Empty. In numeric comparisons, Empty is treated as zero. Since the function then has a zero already, it would then fail to identify the lowest number in the list.