4 CREATION D'UN PROJET

4.1 Généralités

4.1.1 Exe standard

C'est le type de projet le plus courant. Il génère un exécutable 32 bits qui constitue une application. Cette application peut être autonome, ou s'appuyer sur des composants comme les DLL, les OCX, ou les Exe ActiveX.

Une application ou un composant qui utilise des objets fournis par d'autres composants logiciels est désigné sous le nom de **client**. Un client utilise les services d'un composant logiciel en créant des instances des classes fournies par le composant, et en appelant leurs propriétés et leurs méthodes. Dans ce cas, il faut veiller à distribuer l'ensemble des composants nécessaires.

4.1.2 Exe ActiveX

Ce type d'exécutable n'est pas destiné à fonctionner seul. Il s'agit de classes contenant des propriétés et des méthodes qui sont appelées par un Exe standard. Ce type de composant s'exécute dans un autre process de manière asynchrone, c'est-à-dire que le client n'attend pas la réponse du composant pour continuer son exécution.

L'objectif d'un tel projet est de pouvoir exécuter des tâches sans arrêter l'application cliente (multi-tâche).

Ils ont plusieurs paramètres de compilation qui permettent de maintenir la compatibilité avec des versions précédentes du composant. Cette compatibilité permet d'éviter la re-compilation l'Exe client pour que les appels du composant fonctionnent.

4.1.3 DLL ActiveX

Il s'agit de fichiers exécutables qui ne peuvent être lancés seuls. Ils sont obligatoirement appelés par une application tierce. Les DLL sont des modules de classe qui exposent des propriétés et méthodes. Les projets DLL peuvent contenir des modules standards réservés au fonctionnement interne de la classe.
Les DLL offrent l'avantage de pouvoir être appelées simultanément par plusieurs clients. Cela évite la redondance de nombreux services communs à plusieurs application. On peut citer par exemple les boîtes de dialogues « Ouvrir », « Enregistrer », « Imprimer » etc. communes à toutes les applications Office.

Contrairement aux Exe ActiveX, elles s'exécutent dans le même process que l'application cliente. Lors de l'installation d'une DLL sur un poste, elle doit être inscrite dans la base de registre pour être connue du système et exploitée par des clients. L'inscription dans le registre est automatiquement réalisée lors de la compilation.
Si elle n'est pas inscrite dans le registre, elle doit être dans un répertoire de la variable d'environnement « PATH », ou dans le répertoire de l'exécutable.

Les DLL répondent aux mêmes conditions de compilation que les Exe ActiveX.

4.1.4 Contrôle ActiveX

Ce type de projet permet de créer un contrôle personnalisé qui sera compilé en fichier .OCX. Ces fichiers sont inscrits dans le registre comme les DLL. Ainsi, à chaque nouvelle version, il faut supprimer l'ancien dans la base de registre, puis registrer la nouvelle.

Il s'agit de fichiers binaires exploités pour la conception de nouveaux projets sous forme de contrôles ActiveX.

4.2 Les formulaires

Il existe deux types de formulaire : les MDI et SDI. Les formulaires MDI constituent la fenêtre principale d'une application, et contiennent tous les formulaires ayant leur propriété *MDIChild* à *True*. Il ne peut y avoir qu'une seule fenêtre MDI par projet.

Lors de l'exécution, on n'aura qu'un bouton dans la barre des tâches Windows, celui de la MDI. Toutefois, le recours à ce type de fenêtre n'est pas obligatoire.

4.2.1 Chargement, affichage

En dehors du formulaire de démarrage définit dans les propriétés du projet qui s'affiche automatiquement, les autres objets Forms ne sont pas chargés en mémoire. Il est donc nécessaire de gérer le chargement et le déchargement en mémoire ainsi que l'affichage et le masquage des objets Forms.

Par ailleurs, il faut toujours garder à l'esprit que VB est un langage évènementiel. Nous savons que l'utilisateur déclenche des évènements par ces actions sur l'interface (clic sur un bouton), mais le programme lui-même déclenche les évènements d'un objet. Ainsi, lors de l'exécution de la commande : *Charger Form1*, on déclenche l'évènement *Sur Chargement* de l'objet Form1.
Cette imbrication de commandes explicitement définies dans le code, avec les évènements automatiquement déclenchés par VB est très utile, mais il faut bien en tenir compte dans le déroulement de votre programme.

Le code suivant illustre ce concept :
Supposons un Form de démarrage nommé frmDemarre lancé automatiquement par VB et un Form nommé frmClient permettant la saisie d'information dans des zones de texte. Un bouton cmdClient est présent sur notre frmDemarre.

```
Private Sub cmdClient_Click()
        'La commande Load charge le Form frmClient mais ne l'affiche pas à l'écran. L'événement "Load" du
        'frmClient est alors déclenché

        Load frmClient
End Sub
```

Dans ce cas, le frmClient est chargé en mémoire. On peut alors accéder par programme à tous les contrôles qu'il contient et à toutes les variables de niveau module du module

de formulaire associé. Ces objets ne sont pas accessibles lorsque le frmClient n'est pas chargé en mémoire ; votre programme doit en tenir compte.

La procédure qui suit est l'événement *Load* du *frmClient* et elle est automatiquement déclenchée par l'instruction *Load* du code précédent.

```
Private Sub Form_Load()
        'Instructions permettant d'initialiser les contrôles du Form
        frmClient.Show
End Sub
```

La méthode *Show* du *frmClient* permet d'afficher le Form à l'écran. Le *frmClient* est chargé en mémoire à la demande du programmeur par la commande *Load*, ce qui déclenche l'événement Sur Chargement du Form. C'est dans cet événement que le programmeur choisi d'afficher le *frmClient* à l'écran. Cette façon de faire n'est pas impérative mais elle permet de préparer les contrôles de *frmClient*, de rechercher d'éventuelles données dans une base etc. avant que l'utilisateur puisse voir le *frmClient*. Il découvre ainsi le Form parfaitement présenté, sans assister aux traitements réalisés par le programme.

Vous pouvez invoquer directement la méthode *Show* sans passer par *Load*. VB exécute automatiquement le *Load* - donc déclenche l'événement - puis le *Show*. Dans tous les cas, l'événement *Load* du formulaire est exécuté, mais il n'est plus nécessaire d'invoquer la méthode *Show* dans l'événement.

```
Private Sub cmdClient_Click()
        'La méthode Show charge le Form frmClient. L'événement Load du frmClient est alors déclenché.
        frmClient.Show
End Sub
```

4.2.2 Déchargement, masquage

Il est possible de décharger ou de masquer un formulaire selon les mêmes nuances décrites supra.
On trouve une commande *Unload* pour décharger le formulaire de la mémoire, et une méthode *Hide* qui conserve le formulaire en mémoire mais ne l'affiche plus à l'écran.
La méthode *Hide* déclenche l'événement *Deactivate* et la commande *Unload* déclenche l'événement *Unload*. On pourrait envisager l'enchaînement suivant :

Supposons un bouton cmdFermer sur frmClient :

```
Private Sub cmdFermer_Click()
        frmClient.Hide
End Sub

Private Sub Form_Deactivate()
        Unload frmClient
End Sub
```

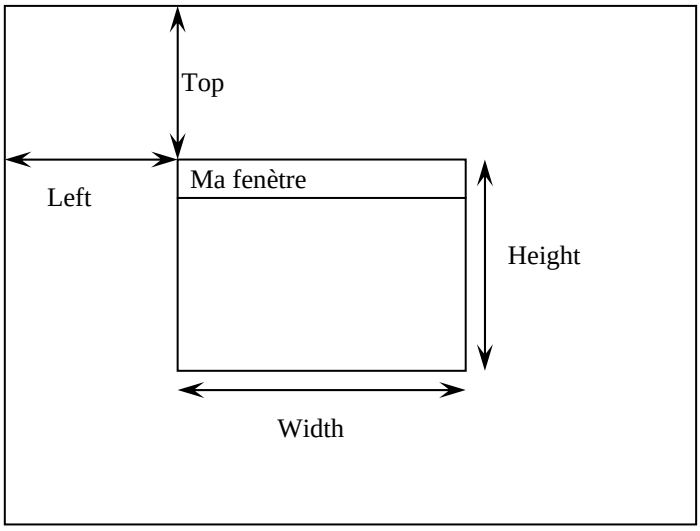
Vous pouvez invoquer directement la commande *Unload*, VB se charge alors de masquer automatiquement le formulaire. Toutefois, les événements sont déclenchés normalement.

4.2.3 Taille et position

Comme beaucoup de contrôles, l'objet Form possède 4 propriétés principales relatives à la taille et la position de l'objet.

Top : coordonnée verticale du coin supérieur gauche par rapport au conteneur.
Left : coordonnée horizontale du coin supérieur gauche par rapport au conteneur.

Height : hauteur du contrôle.
Width : largeur du contrôle.



Lors de la modification des propriétés *Height* ou *Width*, on déclenche l'événement *Resize* de l'objet Form. Le fait de redimensionner la fenêtre en prenant une bordure avec la souris déclenche également l'événement *Resize*.

Il convient de gérer la visibilité des contrôles contenus ainsi que leur taille. En effet, ces derniers restent indifférents au changement de taille du formulaire et peuvent donc se trouver cachés « à l'extérieur » de la zone visible du formulaire. Pour cela, le programmeur gère la taille et la position de tous les contrôles dans l'événement *Resize* du Form, ou bien interdit à l'utilisateur de modifier la taille du Form.

Un formulaire (ainsi que tous les conteneurs) fonctionne comme un repère dont l'origine est le coin supérieur gauche. Ainsi, des valeurs négatives pour les propriétés `Top` et `Left` d'une zone de texte par exemple, la positionne à l'extérieur de la zone visible du formulaire.

4.2.4 Les menus

VB permet d'ajouter des menus aux formulaires, qu'ils soient MDI ou non. La fenêtre MDI porte le menu principal et les fenêtres filles peuvent avoir leur propre menu. Dans ce cas, le menu de la feuille MDI est remplacé par celui de la feuille SDI fille active.

Pour ajouter un menu à un formulaire, il suffit de passer par *Outils / Créateur de menus* ou le raccourci Ctrl+E. Cette action ouvre la fenêtre d'édition des menus. Vous trouvez alors deux propriétés connues *Name* et *Caption*.

Pour chaque entrée du menu, vous devez positionner la propriété *Name*. Elle permet d'accéder au menu dans le code et surtout de programmer son événement *Click()*.
C'est le seul événement disponible pour un menu. Il suffit (en mode création) de cliquer sur le menu pour accéder au module du formulaire et trouver la procédure événementielle : *Private Sub MonMenu_Click().*

Les flèches de la fenêtre de création de menu permettent de définir la position des éléments du menu dans l'arborescence.

On peut également concevoir des menus contextuels. Vous devez créer une branche du menu que vous masquez à l'aide de la propriété *Visible* placée à Faux. Vous appelez ensuite ce menu dans votre code avec la méthode *PopupMenu* selon la structure suivante :

```
object.PopupMenu objMenu
```

Object correspond à un contrôle, souvent un formulaire.
objMenu correspond à un objet menu identifié par sa propriété *Name*.

4.3 Les contrôles et groupes de contrôles

4.3.1 Définition

On appelle contrôles les objets pouvant être dessinés sur un formulaire en mode création. Il s'agit d'objets exposant des propriétés et des méthodes, mais aussi une interface. Toutefois, cette interface n'est pas toujours visible en mode exécution.
On trouve énormément de contrôles, qu'ils soient gratuits ou payants. Certains éditeurs de logiciels vendent des séries de contrôles traitant d'un même thème comme les connexions réseau, les accès aux données, les calculs…
Les contrôles portent l'extension .OCX. On peut trouver des fichiers associés à ces .OCX, comme des .DLL, des .TLB ou .OLB.
Les contrôles disponibles dans l'application sont visibles dans la boîte à outils de VB.

4.3.2 Les principaux contrôles

VB met à disposition quelques contrôles dès l'ouverture de l'environnement de développement. Voici les principaux.

☞ **CommandButton (cmd) :** Il s'agit d'un bouton. Il peut être désactivé ou verrouillé afin de forcer l'utilisateur à faire certaines actions. Il peut contenir une icône ou de la couleur si son *.Style* est positionné à *1-Graphical*. On utilise le plus souvent son événement *Click().*

☞ **Label (lbl) :** Ce contrôle est une étiquette qui permet le plus couramment de mettre des titres à côté d'autres contrôles. L'utilisateur ne peut pas en modifier le contenu *(.Caption).* Il est aussi pratique pour afficher un résultat

par programme ou un message. Il prend également moins de place en mémoire qu'un *TextBox*.

- **TextBox (txt) :** Il s'agit d'une zone de saisie de texte. Il peut également afficher un fichier texte par exemple. On peut le verrouiller contre la saisie par programme, le désactiver (grisé)…

- **OptionButton (opt) :** On l'appelle aussi case d'option ou bouton radio. Il permet de sélectionner une information exclusive parmi d'autres. Lorsque l'on sélectionne une case, on désélectionne celle précédemment activée. Cette exclusion se fait par contrôle conteneur. Si trois *OptionButton* se trouvent sur un formulaire, ils vont être de même niveau et s'excluront les uns et les autres. On peut ajouter 3 autres *OptionButton* dans un contrôle Frame qui est conteneur. Ces derniers seront indépendants des trois premiers. La propriété *Style* permet de choisir des cases radio ou des boutons à bascule.

- **CheckBox (chk) :** Ce sont des cases à cocher. On peut sélectionner autant de cases que l'on veut. La propriété *Style* permet de choisir des cases à cocher ou des boutons enfoncés pour coché ou relâché pour décoché.

- **Frame (fra) :** Il s'agit simplement d'un cadre. Son aspect graphique est simple mais son intérêt vient du fait qu'il est conteneur. De ce fait, si sa propriété *Visible* est placée à *False*, tous les contrôles qu'il contient seront masqués. Si on déplace un contrôle Frame, les contrôles contenus sont aussi déplacés. Il est aussi utile pour les cases d'option.

- **ComboBox (cbo) :** C'est la liste déroulante modifiable. Il se compose d'une zone de texte dans laquelle il est possible de saisir du texte, et d'une partie déroulante destinée à proposer des saisies possibles dans la partie zone de texte. On peut laisser la saisie libre ou imposer que la saisie corresponde à une valeur de la liste.

- **ListBox (lst) :** Ce contrôle affiche une liste de valeur dans laquelle on peut sélectionner une ou plusieurs valeurs selon la propriété *MultiSelect*. Contrairement à la *ComboBox*, la *ListBox* ne propose pas de zone de texte pour la saisie libre.

- **Timer (tmr) :** Ce contrôle n'est pas visible lors de l'exécution. Il dispose d'un seul évènement *Timer*. Cet événement est déclenché à intervalle régulier, et exécute donc du code à chaque expiration de cet intervalle. La valeur est stockée dans la propriété *Interval* et est exprimée en milliseconde. Si *Interval* vaut 5000, le code de l'événement *Timer* sera exécuté toutes les 5 secondes. La propriété *Enabled* permet d'activer ou désactiver l'événement *Timer*.

- **Image (img) :** Ce contrôle permet de stocker une image. Il est plus léger que le contrôle *PictureBox*. Par contre, il n'est pas conteneur d'autres contrôles et ne dispose pas des méthodes avancées du *PictureBox*. Il est souvent utilisé pour faire des boutons personnalisés.

☞ **PictureBox (pic) :** Ce contrôle est plus performant que le contrôle Image en ce qui concerne les techniques graphiques, mais plus consommateur de ressources. Il peut contenir d'autres contrôles et dispose de plusieurs méthodes graphiques.

4.3.3 Manipulation des contrôles

Les contrôles disponibles sont affichés dans la boîte à outils. Pour ajouter un contrôle sur un formulaire, il suffit de cliquer sur son icône dans la boîte à outils et de dessiner une marquise de sélection sur le formulaire.

La fenêtre des propriétés permet ensuite de modifier ses propriétés. **Il est impératif de positionner la propriété** *Name* **de chaque contrôle juste après sa création**. Pour supprimer un contrôle, il faut le sélectionner sur le formulaire et appuyer sur la touche Suppr ou faire un clic droit et sélectionner l'option Supprimer. Le fait de supprimer un contrôle, ne supprime pas le code événementiel qui lui est associé. Ce code devient « orphelin » puisqu'il n'est plus appelé.

Une notion importante est le *Focus*. On dit d'un contrôle qu'il a le focus lorsqu'il est actif. Un contrôle *TextBox* a le focus lorsque le curseur de saisie clignote. Certains contrôles ne manifestent pas si clairement qu'ils ont le focus. Ce concept est important car c'est le contrôle actif qui va répondre aux événements (clavier par exemple). Par ailleurs, les contrôles gérant le focus disposent d'événements associés : *LostFocus* lorsque le contrôle perd le focus et *GotFocus* lorsque le contrôle reçoit le focus.

Un contrôle dans le code est un objet. **Une particularité réside dans le fait qu'ils sont automatiquement instanciés lors du chargement en mémoire du formulaire**. En effet, les objets utilisés en référence (et non en contrôle) doivent être explicitement instanciés (déclaration) dans le code.
En outre, certains objets comme les « common dialog boxes » (comdlg32.dll) peuvent être utilisées sous forme de contrôle ou sous forme de référence.

4.3.4 Les groupes de contrôles

Pour créer un groupe de contrôles, il suffit de copier un contrôle déjà existant puis de le coller sur le même formulaire. VB demande alors à l'utilisateur s'il souhaite créer un groupe de contrôles.
Tous les contrôles d'un groupe ont la même propriété *Name* **et sont distingués par leur propriété** *Index***. Comme les contrôles ont le même nom, ils partagent les mêmes événements.**

Supposons 3 OptionButton faisant partie d'un groupe dont la propriété *Name* serait : *optOpinion*. Les trois cas seraient une réponse à un sondage : oui, non, sans opinion.

On aurait ainsi :

```
optOpinion(0).Caption = « Oui »
optOpinion(1).Caption = « Non »
optOpinion(2).Caption = « Sans opinion »
```

Le chiffre entre parenthèses correspond à l'index du contrôle dans le groupe. Ils ont tous les 3 la propriété *Name* à optO*p*inion. Par contre, leur propriété *Caption* est définie individuellement.

La propriété *Name* décrivant les événements, ces derniers seront communs à tous les contrôles du groupe :

```
Private Sub optOpinion_Click(Index As Integer)

        MsgBox optOpinion(Index).Caption
        'Affiche Oui si Index = 0
        'Affiche Non si Index = 1
        'Affiche Sans opinion si Index = 2

End Sub
```

Remarques :
Dans le cas des groupes de contrôles, le paramètre *Index* est passé aux procédures événementielles pour savoir par quel contrôle du groupe l'événement a été déclenché.
Tous les contrôles d'un groupe sont obligatoirement issus de la même classe (*TextBox*, *Label* etc…)
C'est le seul moyen d'ajouter ou de supprimer des contrôles par programme au moment de l'exécution. Il est nécessaire d'avoir au moins une instance de ce contrôle définie lors de la création, puis on utilise les instructions *Load* et *Unload* pour créer de nouvelles instances.

4.4 Les modules et modules de classe

4.4.1 Les modules standards

Ces modules sont des fichiers textes ayant l'extension **.bas**. Ils peuvent être ajoutés au projet pour contenir des procédures ou fonctions générales. En effet, ces modules ne sont pas associés à un formulaire, ils ne peuvent donc pas contenir de procédures événementielles.

Dans le cas d'un projet n'ayant aucun formulaire (exécution en arrière-plan), vous devez ajouter un module standard contenant la procédure particulière *Sub Main().* Il faut ensuite définir les propriétés du projet pour positionner l'objet de démarrage sur *Sub Main.*

Par ailleurs, les modules standards contiennent les déclarations de portée globale, c'est-à-dire à l'aide du mot réservé *Public*. On y trouve des fonctions ou procédures, des variables ou des constantes. Il est recommandé de déclarer les variables ou constantes globales que dans des modules standards, et non dans les modules de formulaires.

4.4.2 Les modules de classe

Ces modules sont des fichiers textes ayant l'extension **.cls**. Ils permettent de créer des classes personnalisées. La valeur affectée à la propriété *Name* du module de classe déterminera le nom de la classe lors des déclarations appelant cette classe.

Ils répondent aux mêmes critères de portée pour les déclarations, mais ces dernières sont interprétées différemment lors de la création d'instances de cette classe. Des variables publiques dans un module de classe sont considérées comme des propriétés lors de l'instanciation de la classe.

Toutefois, on recourt plutôt à des procédures spécifiques des modules de classe pour accéder aux propriétés : les procédures *Property*. Ces procédures déclarées en portée publique manipulent des variables de portée module. En outre, il est possible d'effectuer des traitements dans ces procédures comme des contrôles de validité etc.

On trouve trois types de procédures propres aux modules de classe :

```
Property Let
Property Set
Property Get
```

Let permet de positionner une propriété d'une classe. *Get* permet de lire une propriété. L'instruction *Set* établit une référence à un objet. En effet, certaines propriétés ne correspondent pas à une valeur (codes couleurs, valeurs numériques…) mais sont des références vers des objets.

Enfin, certains types de projet (dll, ocx…) sont composés impérativement d'un module de classe (avec ajout éventuel de modules standards).

L'essentiel

Ce chapitre concerne essentiellement les éléments de l'IHM. Vous devez maitriser l'utilisation des propriétés et des méthodes des contrôles.

L'IHM est aussi au cœur de l'exécution événementielle. Ce chapitre et le précédent vous ont présenté les éléments de la programmation en VB. Les principes évoqués sont la base de la manipulation des objets.

Vous serez amenés à utilisez beaucoup d'autres contrôles. Chacun d'entres-eux possède sont guide d'utilisation, ses propriétés et méthodes. Cependant, les règles rencontrées jusqu'ici perdurent.