

Les objets et VB

1. Petit rappel sur les objets

D'abord qu'est-ce qu'un objet (pour ceux qui auraient oubliés lol) ? Un objet, c'est un ensemble de données et de traitements sur les données.

Les données sont privées à l'objet, c'est-à-dire que les données ne sont accessibles que de l'intérieur de l'objet. Le seul moyen pour montrer les données au « reste du monde » est d'utiliser les propriétés qui permettent de lire et d'écrire dans les données internes. On lit avec *Property Get* et on écrit avec *Property Let* (ou *Property Set* pour écrire une référence d'objet).

Les traitements sont réalisées par des fonctions ou des procédures qui sont appelées méthodes. On peut définir des méthodes privées pour des traitements internes mais il est nécessaire d'exposer publiquement des méthodes afin de pouvoir donner « une vie » à l'objet.

2. Les objets COM

COM est une modélisation objet. Elle permet de contrôler le cycle de vie de l'objet et repose sur un certain nombre de concepts dont notamment la notion d'interface

- Une interface permet à un objet d'exposer ses fonctionnalités : c'est une liste de propriétés et de méthodes. Il est identifié de façon unique par GUID.

1. Automation

Automation est une technologie permettant à un serveur d'exposer des objets à des clients qui les consomment. Par exemple, Excel expose son objet application afin de pouvoir contrôler des classeurs.

2. Les modules de classe

Un module de classe VB est à la fois la définition de l'interface de l'objet (un typage en quelque sorte) et l'implémentation des fonctionnalités.

Lorsque l'on veut utiliser l'objet, il faut l'instancier à l'aide de la fonction *New*. Cela déclenche le constructeur *Class_Initialize* du module de classe.

Lorsque l'on n'a plus besoin de l'objet, on écrit *Set objet = Nothing*. Cela permet de libérer les ressources de l'objet et déclenche le destructeur *Class_Terminate*.

Le polymorphisme est la faculté d'un objet à exposer une interface différente suivant la demande. Sous VB, il s'implémente avec le mot clé *Implements*. L'objet gèrera son interface plus toutes les interfaces définies par *Implements*.

L'héritage est bien sous VB se n'est rien : VB ne le gère pas dans le langage. Mais derrière la scène tout objet est un héritage de plusieurs interfaces. En résumé, un héritage est un rajout des données et de méthodes. Le père a des données et des méthodes. Le fils, qui hérite du père a les données et les méthodes du père et ses propres données et méthodes. Et ainsi de suite...

Toute méthode a un premier paramètre caché qui est un pointeur vers l'instance de l'objet, donc la structure de l'objet, pour lequel la méthode a été appelée. Sous VB, il s'appelle *Me* et en C++, on l'appelle communément *This*

3. Qu'est ce qu'un objet pour VB

VB est un client Automation qui permet d'utiliser des objets COM et d'en concevoir d'autres.

Un objet pour VB est le type *Object*. Mais qu'est ce que le type *Object* en réalité. Eh bien, c'est un pointeur vers une structure. La seule différence avec une structure classique, c'est le premier membre qui est un pointeur vers un tableau de pointeur vers les méthodes de l'objet, appelé *vtable* (pour *Virtual Function Table*). Les membre suivant sont les données de l'objet. Le type *Object* a à la base, 7 fonctions qui ne lui permettent rien d'autre que d'exister. Nous allons voir tout de suite le détail de ces fonctions.

4. L'interface IUnknown : la notion COM du cycle de vie d'un objet

Cette interface est la base d'un objet COM.

Tout objet supporte l'interface *IUnknown* afin de pouvoir gérer la construction et la destruction d'objets et le référencement des instances de l'objet. Cet interface n'est pas directement le type *Object* mais sert à sa définition par le biais de l'héritage.

Il comporte 3 méthodes qui renvoient toutes un *Long* (en fait un *HRESULT*):

- ***QueryInterface***
- ***AddRef***
- ***Release***

***QueryInterface* a trois paramètre :**

- This qui pointe vers la structure de l'objet
- Refiid est le GUID de l'interface que l'instance de l'objet a créé doit implémenté
- ppvObj est pointe vers un pointeur pour contenir la référence d'un objet de type du GUID

Cette fonction permet de gérer le polymorphisme : un objet peut implémenter plusieurs interfaces.

Si l'objet supporte l'interface, elle renvoie 0 (S_OK) et remplit ppvObj avec l'adresse de l'instance du type correspondant au GUID.

SI l'objet ne supporte pas l'interface, elle renvoie E_NOINTERFACE et 0 dans ppvObj. Elle n'est pas toujours appelée par VB car en générale, elle est lente.

***AddRef* et *Release* ont un seul paramètre : le pointeur This.**

- *AddRef* permet de dire à l'objet que l'on a un pointeur (une référence) de plus vers lui. Cela incrémente un compteur interne qui permet de savoir combien de références de l'objet vivent.
- *Release* permet de dire à l'objet que l'on vient d'éliminer un pointeur (une référence) vers lui. Cela décrémente le compteur interne. Lorsqu'il atteint 0, l'objet est libéré ainsi que ses ressources.

Tout appel à *QueryInterface* et *AddRef* doit être compenser avec un appel à *Release*.

IUnknown permet donc d'implémenter la naissance et la mort de l'objet. Il n'a pas d'équivalent directe dans VB mais est à la base du type *Object*. Bien que VB ne le gère pas

directement en tant que type, il sait très bien l'utiliser dans les interfaces qui ne sont pas *Object* de base.

Beaucoup d'interfaces de Windows dérivent (par héritage) de ce type.

5. L'interface IDispatch

Un pointeur vers ce type est le type *Object = IDispatch.**

Cette interface permet d'exposer des objets vers l'extérieur, comme Excel ou Word, par exemple, sans forcément connaître à l'avance son interface. Il permet de gérer les instances d'objets obtenues par les fonctions *GetObject* et *CreateObject*.

Elle comporte 4 fonctions :

- Les deux premières servent à définir le type de l'objet : ses propriétés, ses méthodes... Ceux sont *GetTypeInfo* et *GetTypeInfoCount*.
- Les deux suivantes servent à l'appel des propriétés et méthodes. Ceux sont *GetIDsOfNames* et *Invoke* que nous détaillerons dans la section Liaison.

Cette interface permet de gérer la liaison tardive (à l'exécution), c'est à dire que les méthodes ne sont pas appelées directement mais uniquement par le biais de cette interface. Cela permet de gérer des noms de méthodes dynamiquement. C'est aussi par cette interface que *CallByName* passe pour les appels de méthodes. Cela permet de ne rien connaître de l'interface de l'objet à la compilation : seul le programmeur connaît les paramètres des méthodes de l'interface. Le compilateur générera le code nécessaire à cette absence d'information.

Cette interface implémente aussi la gestion des événements.

6. Les liaisons

Une liaison est le fait de **relier une méthode ou une propriété à son instance d'objet**. Plus simplement c'est le **point** que vous tapez dans VB : par exemple `App.Path`, le point relie la propriété `Path` à son instance d'objet `App`.

Il existe deux types de liaisons :

- Les liaisons **anticipées** (*vtable binding* et *early-id binding*) effectuées par le compilateur
- Les liaisons **tardives** (*late binding*) effectuées à l'exécution par l'interface *IDispatch*

1) **Le *vtable binding***

Lorsque vous appuyez sur le point, *l'IntelliSense* s'affiche avec la liste des méthodes et propriétés.

Il est utilisé lorsque l'on connaît l'emplacement exacte des méthodes et propriétés de l'interface. **C'est le cas des interfaces ayant une vtable connue à la compilation**, notamment ceux dérivant seulement de *IUnknown*. L'interface fournit toutes les informations nécessaires aux compilateurs pour appeler les méthodes par leur adresse contenues dans la vtable (le premier membre de la structure de l'objet). Il réalise un simple `CALL` (instruction asm) avec l'adresse de la méthode.

Cela donne quelque chose dans le genre de : ptrObj est un pointeur vers l'objet (un pointeur fait 4 octets)

```
MOV EAX,[ptrObj] //met dans EAX l'adresse du début de la vtable
MOV EAX,[EAX + 4 * i] //met dans EAX l'adresse de la i-ème méthode de l'objet
CALL EAX //appel directement la méthode
```

Cette méthode de liaison est très rapide à l'exécution.

2) *Le early-id binding*

Lorsque vous appuyez sur le point, *l'IntelliSense* s'affiche avec la liste des méthodes et propriétés.

Ce type de liaison se produit lorsque l'objet supporte l'interface IDispatch et que l'on connaît l'interface de l'objet mais pas l'emplacement des méthodes dans la vtable. On connaît par contre un identifiant permettant d'utiliser la méthode *Invoke* de l'interface *IDispatch* afin d'appeler la méthode visée.

Le compilateur est donc capable de trouver l'id de la méthode et de passer directement cet id à la méthode *Invoke* (sans passer par un appel à la méthode *GetIDsOfNames*).

Ce type de liaison est déjà beaucoup moins rapide dans la mesure où l'on passe par la méthode *Invoke*.

3) *Le late binding*

Lorsque vous appuyez sur le point, *l'IntelliSense* ne s'affiche pas.

Il correspond au type *As Object* : pour ce type, on ne connaît que sa capacité à gérer *IDispatch* mais pas le type de l'objet.

Avec ce type de liaison le compilateur ne connaît rien sur l'interface de l'objet si ce n'est qu'il supporte l'interface *IDispatch* pour les appels à ses méthodes. C'est le cas lorsque l'on affecte à une variable *As Object* le résultat d'un *CreateObject*. On en sait pas vraiment le type de l'objet que l'on obtient.

On sait que l'on va obtenir une interface de contrôle d'Excel en faisant *Set obj = CreateObject(« Excel.Application »)* mais on ne connaît pas ses méthodes et propriétés puisque l'on n'a pas la bibliothèque de type dans laquelle l'interface est défini.

Un appel à une méthode de l'objet se passe comme suit :

- un appel à la méthode *GetIDsOfNames* qui permet d'obtenir un identifiant pour le nom de la méthode à appeler
- un appel à la méthode *Invoke* permet d'appeler effectivement la méthode ou la propriété en question

Ce mode de liaison est donc très lent.

Conseil : pour réduire le nombre de liaison, utiliser le plus possible With qui fait une copie de la référence de l'objet avant de se lier avec ses méthodes. Le code en sera clarifier et très largement accélérer.

7. Conclusion

D'abord félicitation si vous êtes arrivé jusque là...

Mais, vous allez sûrement me dire : « *Oui, et alors, on en fait quoi de tout ça* ». Eh bien, ce tutoriel n'est qu'une **introduction aux objets COM qu'utilise VB**. Notez aussi le fait que **si vous déclarer une variable IUnknown ou IDispatch, vous ne pourrait pas appeler ses méthodes car VB n'est pas conçu pour vous laisser gérer tout cela**. Et heureusement, car c'est lui qui fait cette partie complexe du travail. Notez enfin, qu'appeler les méthodes ne vous servirait à rien.

Je vous entends déjà dire « *Eh bien alors ca ne sert à rien !* ». A cela, je réponds que cela vous prépare au concept **d'objet « léger »**... Mais cela fait partie du tutoriel suivant...