

Les bibliothèques de types

1. Un début de syntaxe

Une **bibliothèque de types** permet de décrire l'interface de l'objet que vous voulez créer. La description en langage ODL est compilée avec **MkTypLib** pour générer un fichier TLB. Ce fichier est à inclure dans les **références** du projet VB.

a. La syntaxe d'un fichier ODL

Un fichier ODL a la syntaxe suivante :

```
[
    uuid(<GUID>),
    version(<version>)
]
library <Nom de la bibliothèque>
{
    importlib("stdole2.tlb");

    [
        uuid(<GUID>),
        odl
    ]
    interface <Nom de l'interface> : stdole.IUnknown {
        [<attributs>]<valeur de retour> <nom
méthode>(<paramètres>);
    }
}
```

Tout ce qui se trouve entre crochets constitue les attributs de l'entité qui suit.

Le premier <GUID> doit être un GUID unique généré avec **Guidgen**. La version est de la forme *Major.Minor*.

Le second <GUID> (celui de l'interface) doit correspondre au GUID de l'interface, si vous implémenté une interface existante ou un nouveau GUID si votre interface n'existe pas.

Idéalement le nom de l'interface commence par un I. Viennent ensuite, la liste des méthodes qui définit l'ordre dans la vtable.

<attributs> peut comprendre *helpstring*, *vararg*, *propput*, *propputref*, *propget*. La valeur de retour peut être soit **HRESULT**, soit **n'importe quel type sauf un pointeur et il est préférable que ce ne soit pas une structure**. Le nom de la méthode doit être unique dans l'interface (sauf pour les *propput*, *propputref* et *propget*).

<paramètre> à la syntaxe suivante : [<marshaling>]<type de donnée> <nom de paramètre>. <marshaling> peut être soit **[in]**, soit **[in,out]**, soit **[out,retval]** et peut inclure *defaultvalue*(<valeur>).

b. Intégrer une description du contenu

Pour fournir une aide dans *l'Explorateur d'objet de VB*, vous pouvez rajouter entre n'importe quel [] l'attribut *helpstring*(« Description ») **sauf dans les paramètres**.

2. Les Alias

Bien que VB ne puisse pas définir d'alias, il sait très bien les utiliser sauf pour les Enums. Par exemple, on peut définir un alias de long (ou même unsigned long, VB ne le gère pas directement mais peut le gérer à travers un alias. Pour les opérations, ils sont traités comme des signés). Un alias se définit dans un fichier ODL par :

```
Typedef [public] <type de base> <nom de l'alias>
```

3. Les constantes

Les constantes se définissent obligatoirement dans un module. Les constantes se déclarent comme suit :

```
[
    dllname(« n'importe quoi »)
]
module <nom du module>
{
    const <type> <nom constante> = <valeur> ;
}
```

« n'importe quoi » c'est une chaîne quelconque puisque l'attribut *dllname* est obligatoire pour un module.

<type> et <valeur> ne peuvent être que des type simples : *unsigned char, short, long, float* (Single en VB), *double, LPWSTR, LPSTR, BSTR* et c'est tout...

Les chaînes sont définies comme en C : on peut inclure les séquences d'échappement telles que *\n* (nouvelle ligne), *\t* (tabulation), *\r* (retour à la ligne), ** (\)...

4. Les types de données de stdole pour VB

Nom en ODL	Nom en VB	Taille en octets
<i>unsigned char</i>	Byte	1
<i>Short</i>	Integer	2
<i>Long</i>	Long	4
<i>Float</i>	Single	4
<i>Double</i>	Double	8
<i>currency</i>	Currency	8
<i>DATE</i>	Date	8
<i>BSTR</i>	String	4 + 2 * nombre de caractères + 2
<i>IDispatch*</i>	Object	4
<i>boolean</i>	Boolean	2
<i>VARIANT</i>	Variant	16 au moins

<i>SAFEARRAY(<type>)*</i>	Tableau en paramètres	4
<i>Struct</i>	Type/End Type	Somme de la taille des members
<i>SAFEARRAY(<type>)</i>	Tableau de taille variable	24
<i><type> <nom>[<taille>]</i>	Tableau fixe	Sizeof(<type>) * <taille>
<i>Enum</i>	Enum	4
<i>LPSTR</i>	<i>Pas d'équivalent</i>	Nombre de caractères + 1 (ANSI)
<i>LPWSTR</i>	<i>Pas d'équivalent</i>	2 * Nombre de caractères + 2 (UNICODE)

5. Les structures

Dans le langage ODL, une structure se définit comme suit :

```
typedef struct {
    <type> <nom> ;
    ...
} <nom structure> ;
```

Dans une structure, on ne peut inclure que les types suivants :

- **Les types simples** : *long, float, DATE, ...*
- **Les types plus complexes** :
 - **Les tableaux** : *SAFEARRAY(<type>)*
 - **Les Variant** : *VARIANT*
 - **Les chaînes** : *BSTR* mais pas *LPSTR, LPWSTR*
 - **Les tableaux de taille fixe** : *<type> <nom>[<taille>]*
 - **Les objets** : *<interface>**

A noter que l'on ne peut pas définir un typedef struct qui porte le nom de GUID et surement d'autres.

A noter aussi que l'on peut régler l'alignement des données afin d'obtenir un alignement différents du 4 octets de VB. Sous *mktyplib*, cela se fait sur la ligne de commande */align <valeur>*.

6. Les modules : une autre façon de déclarer des APIs

Un module permet aussi de déclarer des fonctions de DLLs.

La syntaxe est la suivante :

```
[
    dllname (« <nom de la dll> »)
]
module <Nom du module>
{
    [entry(<index ou nom>)] <type> <nom de
fonction>(<paramètres>) ;
    ...
}
```

<index ou nom> est soit le nom de la fonction, soit sont index. <nom de la dll> est le nom de la dll (avec ou sans chemin).

A noter, une restriction dans les attributs des paramètres : **on ne peut pas utiliser retval**. Nous sommes donc limité à [in] et [in,out].

7. Ce que VB ne sait pas gérer dans les typelibs

Tout ce qui va être définit ici **doit être remplacé pour être utilisé par VB** :

- **Les paramètres « tableau par valeur ».** VB ne sait gérer que les tableaux passés par référence (SAFEARRAY(<type>)* en ODL).
- **Les paramètres « structures par valeur ».** VB ne sait gérer les structures passés par référence (<structure>* en ODL). Il faut déclarer autant de paramètre de type *Long* que de champs dans la structure et ainsi remplir la pile avec les champs de la structure
- **Les pointeurs dans les structures.** Il suffit de les remplacer par des long.
- **Les types non signés.** VB ne gère comme type *non signé* que Byte qui est un *unsigned char*. VB ne sait gérer ni *char* (ne pas confondre avec char*) ni *int*. Il suffit de remplacer tous les *unsigned long* ou *short* par *long* et *short* et, *char* par *unsigned char*.
- **Les chaînes de caractères de taille fixe dans une structure.** Il faut le remplacer par un tableau de *short* <nom>[<taille>].
- **Les unions.**
- **Les paramètres de type [out] <type>***. Il peut être remplacé par [in,out]<type>* si l'on passe toujours une variable ou par [in] long si l'on est amené à passer un *NULL* (0). On peut enfin le transformer en [out,retval] si c'est le dernier paramètre. Il ne peut y avoir qu'un seul [out,retval]. Il devient alors la valeur de retour de la fonction. Voir plus loin pour les valeurs de retour.

8. VB-izé une interface

Pour trouver une interface déjà définie, vous pouvez utiliser l'utilitaire *OleView.exe* et récupérer son *IID* (*GUID*).

- **Vérifié que vous N'avez PAS les attributs oleautomation et dual dans les attributs des interfaces. Ajoutez l'attribut odl.**
- Ajouter en tête toutes les fonctions des interfaces dont l'interface hérite sauf *IUnknown*. L'interface doit dériver de *IUnknown*. Sa déclaration doit être : interface <nom de l'interface> : stdole.IUnknown.
- Remplacer tout ce que VB ne supporte pas par les équivalents (voir section « Ce que VB ne sait pas gérer dans les typelibs »)
- Changer tout ce qu'il faut pour que VB arrête de faire le difficile

9. Les différents type de paramètres pour les fonctions

Tous les types de bases : **Byte**, **Integer**, **Long**, **Single**, **Double**, **Boolean**, **Currency**, **Date**, **Enum** sont des types qui peuvent être passé directement *par valeur*. Pour ces types, si le paramètre est défini :

- **[in] <type>** , c'est un **ByVal As <type>**
- **[in,out] <type>***, c'est un **ByRef As <type>**

<type> est à choisir parmi un nom d'*enum*, *unsigned char*, *short*, *long*, *float*, *double*, *boolean*, *currency*, *DATE*.

Il n'y pas d'autres combinaisons. A noter que pour ces types, on peut définir une valeur par défaut, qui rend le paramètre optionnel, avec l'attribut *defaultvalue(<valeur>)*.

Les types **String** (**BSTR**, **LPSTR**, **LPWSTR**), **Object** (**IDispatch***) et autres types *objets* (toutes les interfaces) sont, *par définition*, des types *pointeurs*. Pour ces types, on définit :

- **[in] BSTR**, **[in] LPSTR**, **[in] LPWSTR**, c'est un **ByVal As String**
- **[in,out] BSTR***, **[in,out] LPSTR***, **[in,out] LPWSTR***, c'est un **ByRef As String**
- **[in] IDispatch***, c'est un **ByVal As Object**
- **[in,out] IDispatch****, c'est un **ByRef As Object**

A noter que **ByRef As Object** (ou **As <interface>**) n'est nécessaire que lorsque l'on compte modifier la référence d'objet, ou en renvoyer une. Pour un simple passage de paramètre à des fins d'utilisation, on optera toujours pour un **ByVal**

Les types tableaux sont eux aussi des pointeurs :

- **[in] SAFEARRAY(<type>)*** et **[in,out] SAFEARRAY(<type>)*** sont équivalent puisque l'on ne peut pas passer de tableau par valeur.

<type> est n'importe quel type, *simple*, *structure* ou *tableau*.

10. Les paramètres [out,retval]

Le dernier paramètre de la liste peut être attribué avec **[out,retval]** s'il est du type **ByRef** c'est à dire **pointeur** (*long**, *boolean**, ..., *BSTR**, *IDispatch***, *<interface>***, ...). **Il ne peut y en avoir qu'un seul.**

11. Les ParamArray : l'attribut vararg

Pour déclarer que la liste des paramètres n'est pas connue à partir d'un certain paramètre, on utilise **ParamArray** suivi d'un nom de paramètre de type tableau de **Variant** (sous VB). En langage ODL, il faut ajouter **vararg** dans les attributs de la méthode (et non dans les attributs du paramètre). Il s'en suit une définition de méthode comme suit :

```
[vararg] <type> <nom>(<liste de paramètres connus>, SAFEARRAY(VARIANT) * <nom arg>) ;
```

12. Les propriétés : les attributs *propput*, *propputref* et *propget*

Pour déclarer une propriété en *lecture-écriture*, il faut *deux fonctions* : une pour *lire* et une pour *écrire*. Nous distinguerons deux cas :

- Les propriétés pour types normaux

- Les propriétés pour types objets

1. Les propriétés normales : *Property Get / Property Let (propget / propput)*

Une propriété de ce type aura deux entrées dans la vtable. En langage ODL, elle auront le prototype suivant :

- Pour la lecture, Property Get :
 - **[propget] HRESULT <nom propriété>([out,retval] <type>* <nom>);**
- Pour l'écriture, Property Let :
 - **[propput] HRESULT <nom propriété>([in] <type> <nom>);**

2. Les propriétés objets : *Property Get / Property Set (propget / propputref)*

Une propriété de ce type aura deux entrées dans la vtable. En langage ODL, elle auront le prototype suivant :

- Pour la lecture, Property Get :
 - **[propget] HRESULT <nom propriété>([out,retval] <interface>** <nom>);**
- Pour l'écriture, Property Let :
 - **[propputref] HRESULT <nom propriété>([in] <interface>* <nom>);**

On peut aussi avoir des propriétés *en lecture-seule ou écriture-seule* en supprimant une des deux fonctions.