INSTITUT GEOGRAPHIQUE NATIONAL

E N S G

École Nationale des Sciences Géographiques

Introduction à la programmation en VBA sur ArcGIS

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES
Centre d' Etudes et de Recherches en SIG (CERSIG)
6 et 8 avenue Blaise Pascal - Cité Descartes - Champs-sur-Marne - 77455 MARNE LA VALLÉE CEDEX 2
Télephone 01 64 15 31 83 Télécopie 01 64 15 31 07 - maill : Nathalie.Sillard@ensg.ign.fr

Sommaire

1. Introduction

ArcInfo fournit un kit de développement **"ArcObjects"**, plate-forme de développement de la famille des applications ARCGIS tels que ArcMap, ArcCatalog...
Plusieurs types de développements sont possibles :

- Personnaliser l'environnement dans ArcMap ou ArcCatalog avec **VBA** (Visual Basic for Application), en développant des **macros** associées à un document ArcMap (*.mxd) ou un modèle (*.mxt).
- Créer des applications ou des contrôles, en développant en **VB, C++, Delphi**..., des DLL, des contrôles ActiveX ou des exécutables.
- Etendre le modèle ArcInfo grâce à Visio-UML et les outils CASE, avec les langages C++, Delphi ou tout autre langage supportant la technologie COM (Component Object Model).

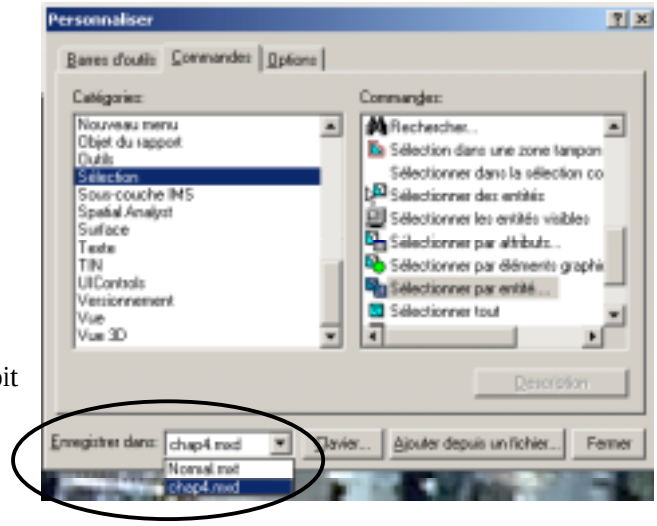
Ce manuel décrit plus spécifiquement l'utilisation d'ArcObjetcs avec **VBA** pour la personnalisation d'ArcGIS. Visual Basic for Application est à la fois un langage et un environnement de développement. C'est un langage interprété dont l'intérêt est d'être partagé également par d'autres applications (notamment bureautiques : Word, Access, Excel ...).

Ce cours s'adresse aux débutants en programmation sur ArcGIS ayant un pré-requis en programmation (VBA ou autre langage). Après un chapitre (chap. 2) consacré à la personnalisation de l'interface ArcMap sans programmation, le chapitre 3 propose un bref rappel de la programmation en VBA. Le chapitre 4, essentiel pour la compréhension de ce cours, décrit les principes de la programmation sur ArcGIS (description du vocabulaire, comment utiliser les diagrammes, qu'est-ce que la norme COM etc ...). Les 13 chapitres suivants décrivent plus en détail les différentes possibilités de développement suivant les thèmes souhaités (analyse, mise en page, symbolisation, gestion des données etc ...) et sont illustrés d'exemples de code (`sur fond grisé, police courrier`)

2. Personnalisation de l'interface : introduction

2.1. LA BOITE PERSONNALISER

La boîte de dialogue "Personnaliser" permet de gérer les barres d'outils et commandes (boutons, menus) d'ArcMap ou ArcCatalog. Par exemple, on peut ajouter ou supprimer des commandes par "glisser-déposer", les déplacer, modifier leur aspect, réorganiser les menus etc ...



Les personnalisations sont enregistrées soit dans le document courant, soit dans le modèle Normal.mxt (elles seront valables dans tous les documents ArcMap).
cf. § ci-dessous.

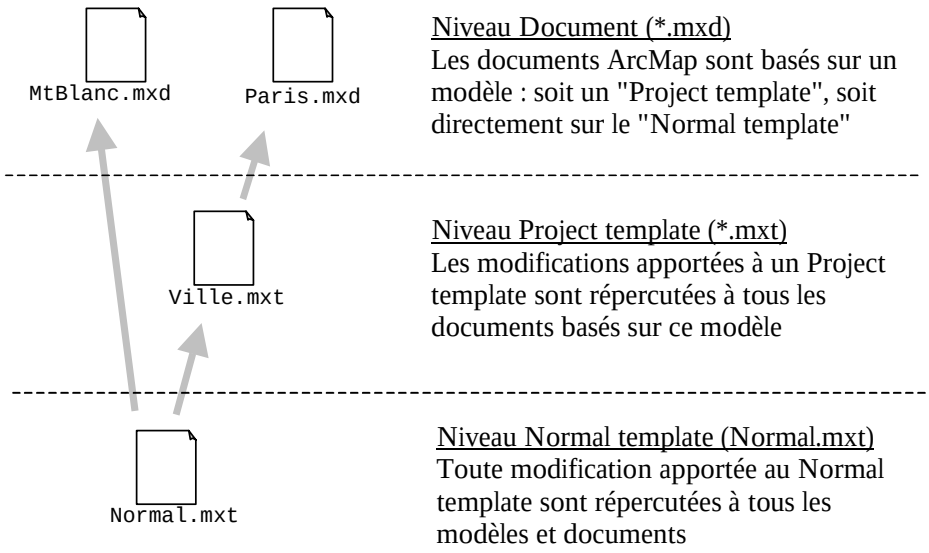
Par exemple, pour créer la nouvelle barre d'outil :

il suffit d'aller dans l'onglet "Barres d'outils" puis appuyer sur le bouton "Nouvelle", nommer la nouvelle barre :

puis, à partir de l'onglet "Commandes", choisir la catégories de la commande à ajouter et glisser/déposer la (ou les) commandes souhaitées dans la boîte.

2.2. SAUVEGARDE DES PERSONNALISATIONS

Les modifications faites dans la fenêtre "Personnaliser" ou bien le code écrit en VBA pour personnaliser l'interface ArcMap peuvent être enregistrés à divers endroits sur le disque, correspondant aux trois niveaux possibles de modèles et documents ArcMap :



Niveau Document (*.mxd)
Les documents ArcMap sont basés sur un modèle : soit un "Project template", soit directement sur le "Normal template"

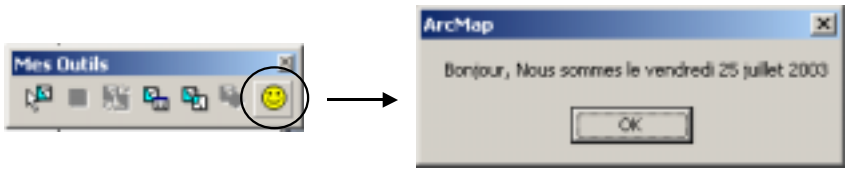
Niveau Project template (*.mxt)
Les modifications apportées à un Project template sont répercutées à tous les documents basés sur ce modèle

Niveau Normal template (Normal.mxt)
Toute modification apportée au Normal template sont répercutées à tous les modèles et documents

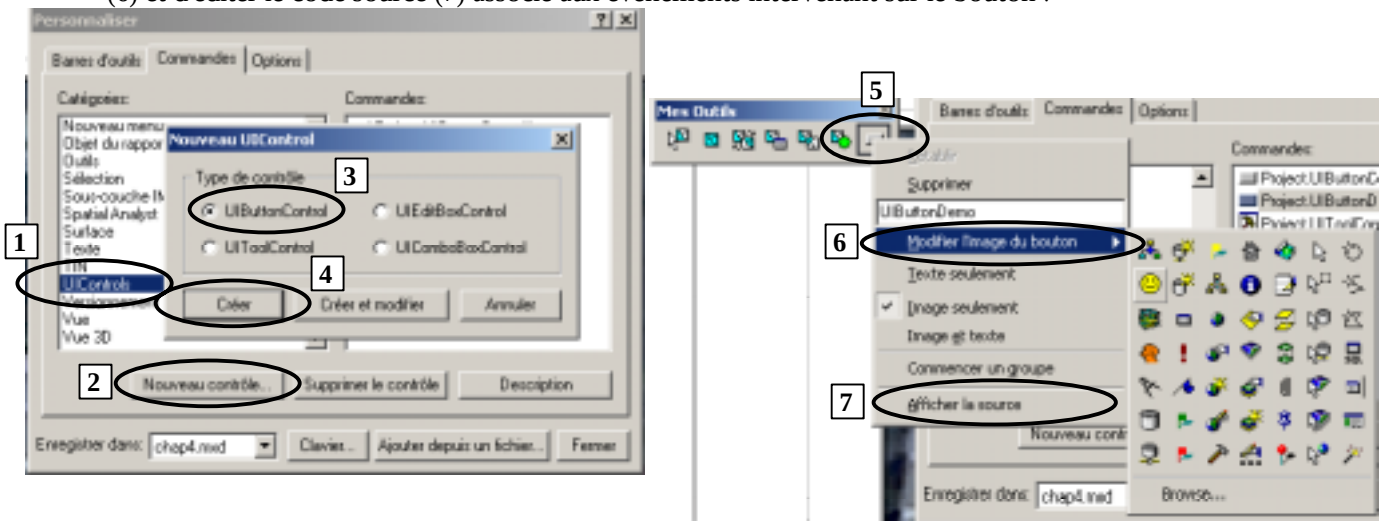
En ce qui concerne la personnalisation d'ArcCatalog, tout est stocké dans le "Normal.gxt", puisqu'il n'existe pas la notion de document avec ArcCatalog.

2.3. EXEMPLE DE CREATION D'UNE COMMANDE

L'exemple ci-dessous montre comment créer un bouton dans la boîte d'outil décrite précédemment qui lorsqu'on clique dessus fait apparaître un petit message de bienvenue :



La première étape consiste à créer un nouveau "contrôle" (c'est à dire une commande) en choisissant l'item "UIControl" dans la liste des commandes (1), puis "Nouveau contrôle (2). On choisit ensuite de créer un nouveau "UIButtonControl" (3 et 4). Il suffit ensuite de déplacer la commande ainsi créée dans la boîte à outil souhaitée. Un clic-droit sur l'outil (5) permet de modifier l'image du bouton (6) et d'éditer le code source (7) associé aux événements intervenant sur le bouton :



Il reste à implémenter la ligne de code permettant d'afficher le message lors d'un événement clic sur le bouton (procédure "<Nom_du_bouton>_Click"), par exemple :

```
Private Sub UIButtonDemo_Click()
  MsgBox "Bonjour, Nous sommes le " & Format(Date, "dddd d mmmm yyyy")
End Sub
```

De même, il est possible d'implémenter le code nécessaire pour créer un "tooltip" (petit message qui apparait sur fond jaune lorsqu'on passe avec la souris sur le bouton) grâce à la fonction événementielle suivante :

```
Private Function UIButtonDemo_ToolTip() As String
  UIButtonDemo_ToolTip = "Date du jour"
End Function
```

3. Introduction à Visual Basic for Application

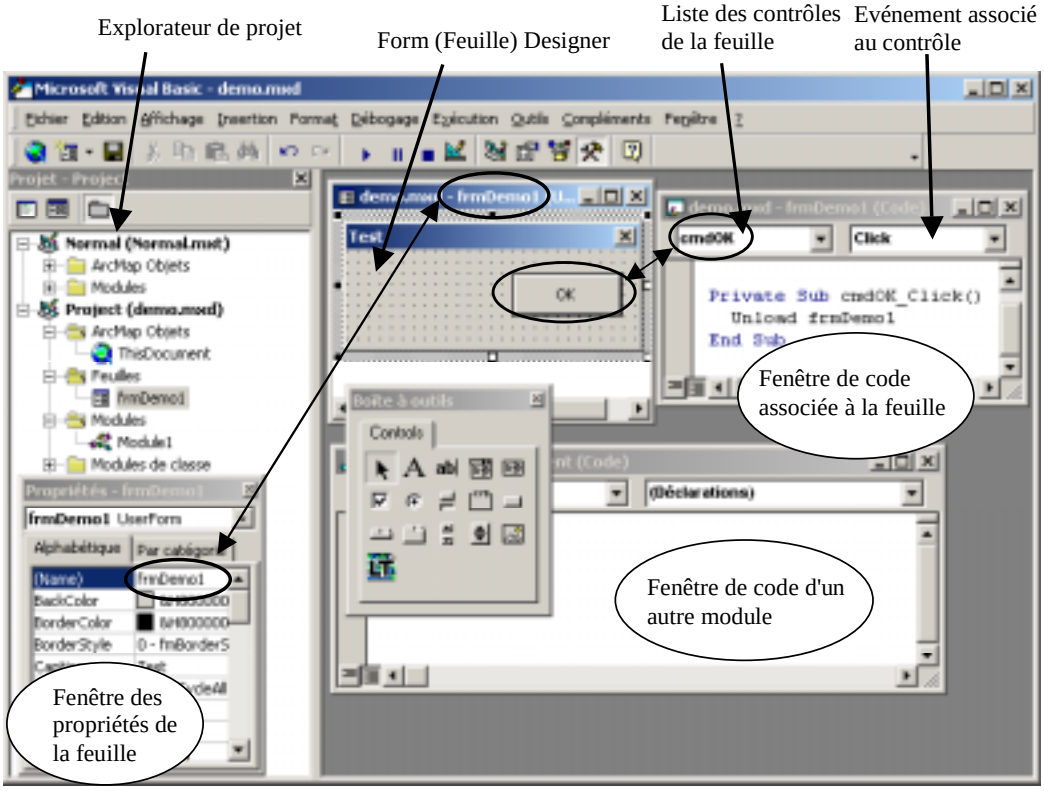
Ce chapitre est un bref rappel de l'environnement VBA et de la syntaxe du langage. Pour davantage de précisions, se reporter au cours ENSG de la CPRI : "Visual Basic 6.0"

3.1. EDITEUR VISUAL BASIC

La fenêtre de projet liste tous les modules de code de la carte courante, du Normal template ou bien d'un autre modèle sur lequel la carte est basée. Chaque module contient un certain nombre de "macros"(ou procédures) elles-mêmes constituées d'un certain nombre de lignes de code (instructions). Il existe trois type de modules :
- les modules standards
- les feuilles (ou "Userform")
- les modules de classe (cf. chap. 1)

Une feuille est une boîte de dialogue créée par le développeur, contenant un certain nombre de "contrôles", par exemple des boutons, listbox, etc ... A ces contrôles sont associés des événements associés à des procédures événementielles (par exemple, à l'événement clic sur un bouton, on va déclencher un certain traitement).
Il est préférable, pour une meilleure lisibilité du programme d'utiliser une convention de dénomination des contrôles, par exemple :

Type de contrôle	Préfixe	Exemple de nom
CommandButton	cmd	cmdOK
Form	frm	frmDemo
Label	lbl	lblX
Textbox	txt	txtPrenom



Remarque : Lorsqu'on utilise l'éditeur VBA avec ArcMap, il existe un module "ThisDocument" gérant les événements liés au document (ouverture, fermeture etc ... cf exemple § 5.5)

3.2. SYNTAXE VBA : RAPPELS

3.2.1. Types de variables

Il est préférable, pour une meilleure lisibilité du code, d'utiliser une convention faisant précéder chaque nom de variables d'un préfixe dépendant du type de variable, par exemple :

Type (taille en octets)	exemple de **valeur**	**Prefix**	exemple de **nom**
String (10 + length)	"toto"	str	strNom
Boolean(2)	True ou False	bln	blnFlag
Date(8)	de 1/1/100 à 12/31/9999	dat	datNaissance
Byte(1)	0 à 255	byt	bytAge
Integer(2)	-32768 à 32767	int	intNum
Long Integer(4)	-2147483648 à 2147483647 (2^{31})	l	lNum
Single(4)	de -3,402823E38 à -1,401298E-45 (valeurs négatives) et de 1,401298E-45 à 3,402823E38(valeurs positives)		
Double(8)	de -1,79769313486231E308 à -4,94065645841247E-324 et de 4,94065645841247E-324 à 1,79769313486232E308	dbl	dblSurface
Variant(>16)	Type de données pouvant contenir des données de toutes sortes	var	varLongitude

La ligne "`Option Explicit`" en tête de module oblige le développeur à déclarer toutes les variables. En son absence, c'est le type de données "Variant" qui est attribué à toute variable non déclarée.

3.2.2. Déclaration et portée

Il existe trois niveaux de déclaration d'une variable :

- Niveau procédure :

La variable est uniquement valable à l'intérieur de la procédure. Elles est détruite quand la procédure est terminée. On utilise les mots-clé **Private** ou **Dim** pour déclarer la variable dans la procédure, par exemple :

```
Private Sub Exemple()
  Dim i as integer
  i = i + 1
  MsgBox i
End Sub
```

Dim ré-initialise les variables (dans l'exemple ci-dessus, i = 0 après la déclaration). Pour conserver la valeur de la variable entre chaque éxecution de la procédure, il faut utiliser le mot-clé "**Static**". Dans l'exemple suivant, la variable i est incrémentée à chaque exécution de la procédure :

```
Private Sub Exemple()
  Static i as integer
  i = i + 1
  MsgBox i
End Sub
```

- Niveau module :
La variable est valable à l'intérieur du module tout entier. Elle est déclarée dans l'entête du module avec **Dim** ou **Private** puis on l'initialise dans la procédure, par exemple :

```
Dim strNom as string

Private Sub Exemple()
  strNom = "Toto"
  ' traitements ....
End Sub
```

- Niveau Public :
La variable est valable dans tout le projet. On utilise par exemple ce type de variable lorsqu'on manipule plusieurs "forms". Elle est déclarée avec le mot-clé "Public". Si la variable est déclarée dans une "form", il faut faire référence à celle-ci lorsqu'on l'utilise ailleurs, par exemple :

```
form2.strNom = "Toto"
```

Dans les gros projets, il est fréquent de créer un module spécial pour la déclaration des variables "Public".

Remarques :

- Attention à la façon de déclarer les variables :

```
Dim count as Long
Dim max as Long
```

est équivalent à

```
Dim count as Long, max as long 'moins lisible
```

mais par contre, dans la ligne suivante, count est défini comme "variant" :

```
Dim count, max as Long
```

- Un tableau est une variable simple comportant de nombreux compartiments permettant de stocker autant de valeurs, par opposition à une variable classique dotée d'un compartiment destiné à une seule valeur. La déclaration d'un tableau se fait comme suit :

```
Dim curExpense(364) As Currency
```

qui s'utilise ensuite de la manière suivante :

```
Sub FillArray()
    Dim curExpense(364) As Currency
    Dim intI As Integer
    For intI = 0 to 364
        curExpense(intI) = 20
    Next
End Sub
```

- Les constantes se déclarent comme suit (elles sont de type Private par défaut) :

```
Const MyVar = 459
```

ou bien :

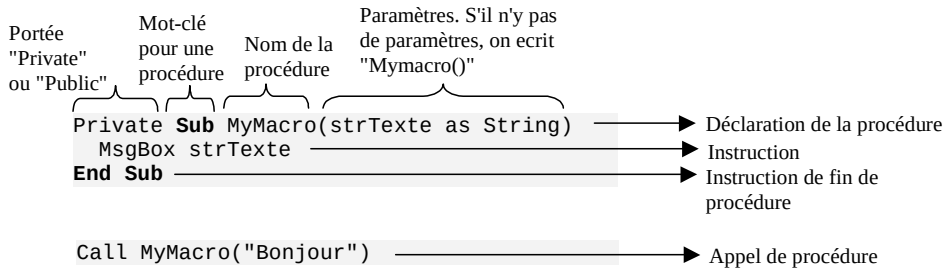
```
Public Const MyPublicVar = 5784
```

3.2.3. Les différents types de procédures

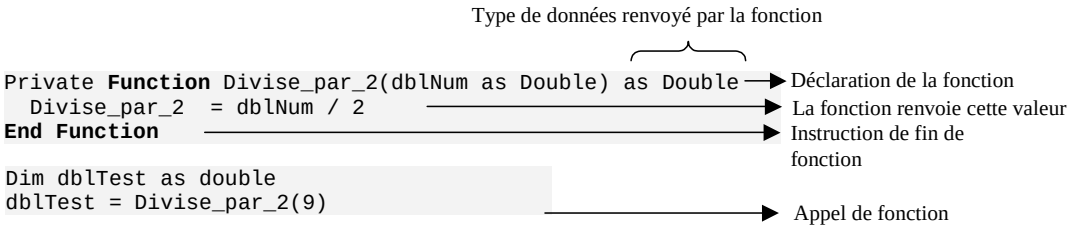
- Procédures événementielles :
Elles sont associées à un objet (et sont exécutées quand l'événement correspondant se réalise. La syntaxe est la suivante : "object_event(parameters)". Certains événements n'ont pas de paramètres (clic sur un bouton par exemple), alors que d'autres utilisent plusieurs paramètres (MouseMove par exemple) :

```
Private Sub CmdButton1_Click() 'Clic sur le bouton CmdButton1
  ' ............
End Sub
Private Sub UIToolControl1_MouseMove(ByVal button As Long, _
              ByVal shift As Long, ByVal x As Long, ByVal y As Long)
  ' Déplacement avec la souris de l'outil UIToolControl1
 '..........
End Sub
```

- Procédures ou "SubRoutines" :
Une procédure est une série d'instructions exécutant des actions mais ne renvoyant pas de valeurs. Contrairement à une procédure événementielle, elle doit être appelée. La syntaxe est la suivante :



- Fonctions :
Une fonction est similaire à une procédure mais elle renvoie une valeur en affectant une valeur à son nom dans une ou plusieurs instructions de la procédure. La syntaxe est la suivante :



Si la valeur renvoyée par une fonction n'est pas utilisée, on peut appeler la fonction de la même manière qu'une procédure, sans affecter la fonction à une variable et en supprimant les parenthèses, par exemple :

```
MsgBox "Traitement terminé", vbExclamation, "Calcul des intersections ..."
```

Attention, l'utilisation de parenthèses dans l'exemple précédent renvoie un message d'erreur :

```
MsgBox ("Traitement terminé", vbExclamation, "Calcul des intersections ...")
```

- Sorties de procédures :
Exit Sub permet de sortir d'une procédure
Exit Function permet de sortir d'une fonction

3.2.4. Instructions conditionnelles

- "If...Then...Else" :
L'instruction **If...Then...Else** permet d'exécuter une instruction spécifique ou un bloc d'instructions, selon la valeur d'une condition. La syntaxe est la suivante :

```
If condition Then
  [statements]
[Else elsestatements]
```

Ou bien

```
If condition Then
  [statements]
[ElseIf condition-n Then
  [elseifstatements] ...
[Else
  [elsestatements]]
End If
```

Par exemple :

```
Sub AlertUser(value as Long)
    If value = 0 Then
        AlertLabel.ForeColor = vbRed
        AlertLabel.Font.Bold = True
        AlertLabel.Font.Italic = True
    Else
        AlertLabel.Forecolor = vbBlack
        AlertLabel.Font.Bold = False
        AlertLabel.Font.Italic = False
    End If
End Sub
```

- "Select Case" :

L'instruction **Select Case** s'utilise en en remplacement de **ElseIf** dans des instructions **If...Then...Else** lors de la comparaison d'une expression à plusieurs valeurs différentes. La syntaxe est la suivante :

```
Select Case testexpression
  [Case expressionlist-n
    [statements-n]] ...
  [Case Else
    [elsestatements]]
End Select
```

Par exemple :

```
 Function Bonus(performance, salary)
     Select Case performance
         Case 1
             Bonus = salary * 0.1
         Case 2, 3
             Bonus = salary * 0.09
         Case 4 To 6
             Bonus = salary * 0.07
         Case Is > 8
             Bonus = 100
         Case Else
             Bonus = 0
     End Select
 End Function
```

3.2.5. Boucles

- "For..Next" :

Les instructions **For...Next** permettent de répéter un bloc d'instructions un certain nombre de fois. Les boucles **For** utilisent une variable de compteur dont la valeur est incrémentée ou décrémentée à chaque itération (suivant la valeur définie par le mot-clé step), par exemple :

```
Sub TwosTotal()
    For j = 2 To 10 Step 2
        total = total + j
    Next j
    MsgBox "Le total est de " & total
End Sub
```

- "Do..Loop" :
Les instructions **Do...Loop** permettent d'exécuter un bloc d'instructions un nombre de fois indéfini, tant qu'une condition a la valeur **True** (avec while) ou jusqu'à ce qu'elle prenne la valeur **True** (avec until) :

```
Do [{While | Until} condition]
  [statements]
Loop
```

- Sorties de boucles :
Exit Next permet de sortir d'une boucle For..Next
Exit Do permet de sortir d'une boucle Do..Loop

3.2.6. Passage d'arguments : ByRef et ByVal

Tous les arguments sont passés aux procédures par référence sauf indication contraire. Cette méthode est efficace en ce sens que le temps de passage des arguments et l'espace mémoire qu'ils occupent dans une procédure (4 octets) sont les mêmes quel que soit le type de données de l'argument.
Il est également possible de passer un argument par valeur en incluant le mot clé **ByVal** dans la déclaration de la procédure. Un argument passé par valeur occupe de 2 à 16 octets dans la procédure, en fonction de son type de données. Les types de données plus importants impliquent un temps de traitement supérieur à celui des types de données moins volumineux. Les types de données **String** et **Variant** ne doivent donc généralement pas être passés par valeur.
Lorsqu'un argument est passé par valeur, la variable d'origine est copiée dans la procédure. Les modifications apportées ensuite à l'argument au sein de la procédure ne sont pas répercutées sur la variable d'origine.

3.2.7. Quelques remarques supplémentaires

Il est possible d'écrire une instruction sur plusieurs lignes grâce au caractère " _" (espace suivi de blanc souligné) :

```
strNameList = "La carte contient " & pMap.LayerCount _
              & "couches :" & vbCrLf
```

est équivalent à

```
strNameList = "La carte contient " & pMap.LayerCount & "couches :" & vbCrLf
```

L'instruction **With** permet de spécifier un objet pour une série d'instructions. Les instructions **With** accélèrent l'exécution des procédures et permettent d'éviter des saisies répétitives.

```
Sub FormatRange()
    With Worksheets("Sheet1").Range("A1:C10")
        .Value = 30
        .Font.Bold = True
        .Interior.Color = RGB(255, 255, 0)
    End With
End Sub
```

4. Introduction à ArcObjects

ArcObjects s'appuie sur la technologie **COM (Microsoft's Component Object Model)**, il est donc possible d'étendre l'architecture d'ArcObjects en développant des composants utilisant la norme COM, de la même manière que peuvent le faire les développeurs d'ESRI. Attention, COM n'est pas un langage orienté objet mais une norme définissant des protocoles de connections entre différents composants logiciels, indépendamment du langage de programmation. Il est ainsi possible de développer des composants logiciels réutilisables et échangeables. COM définit également un modèle de programmation dit "**Interface**-based programming".

Les ArcObjects OMD (Object Model Diagrams) utilisent la modélisation UML avec quelques variantes afin de pouvoir représenter les spécificités de la norme COM (notament les interfaces). Ces modifications ajoutent des symboles pour représenter les différents types de propriétés et méthodes, représenter les relations d'instantiation et marquer la différence entre abstract classes, coclasses et classes (cf. § ci-dessous). Ces diagrammes aident à comprendre l'architecture d'ArcObjects et complètent l'information accessibles via l'environnement de développement (liste des objets, méthodes et propriétés).

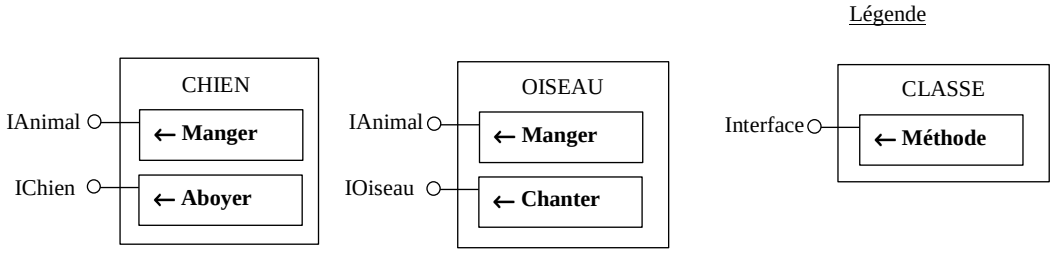
4.1. COM ET INTERFACES

4.1.1. Pourquoi des interfaces ?

Développer avec COM implique développer en utilisant des **interfaces**. Une interface est un groupe logique de méthodes et propriétés. Pour communiquer avec un objet, on passe par son interface.
Les interfaces permettent de faire évoluer une classe en ajoutant une nouvelle interface. Une interface ne peut jamais être supprimée mais l'implémentation peut être modifiée. Certains langages orienté-objet utilisent des classes et des objets mais pas d'interfaces, ce qui peut poser problème lorsqu'une classe a besoin d'être mise à jour. En effet, lorsqu'une classe et le code associé évoluent, les développements utilisant cette classe risquent de devenir obsolètes, voire ne plus fonctionner.
Les interfaces résolvent ce problème, puisque lorsqu'une classe a besoin d'être re-programmée, une nouvelle interface est créée.

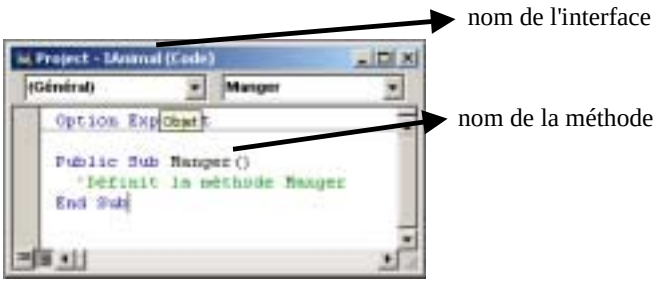
4.1.2. Implémentation

Prenons l'exemple suivant :

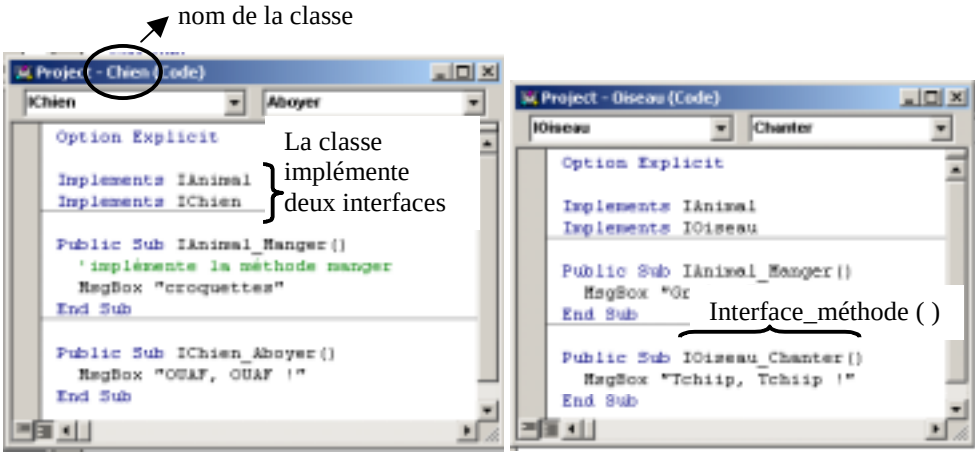


La classe "CHIEN" implémente deux interfaces "IAnimal" (comportant la méthode "Manger") et "IChien" (comportant la méthode "Aboyer"). De même, la classe "OISEAU" implémente deux interfaces "IAnimal" et "IOiseau".

Pour chaque interface un module de classe dans le projet VBA définit les propriétés et méthodes (pas de code dans les procédures) :

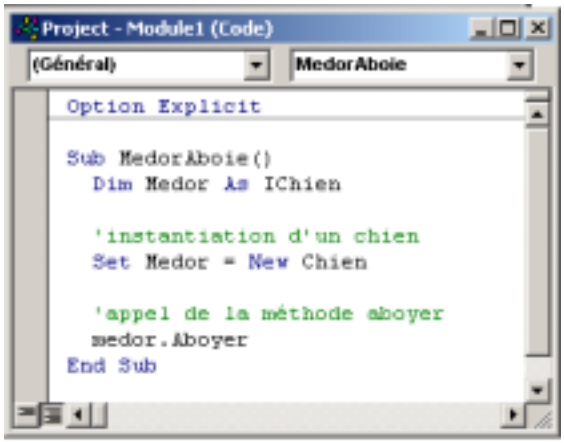


Pour chaque classe, le module de classe liste les interfaces implémentées par la classe dans la section des déclarations (Mot-clé **Implements**). Pour chaque interface, le code définit les propriétés et méthodes :



Plusieurs modules de classe peuvent implémenter la même interface avec un code différent : c'est le **polymorphisme**.

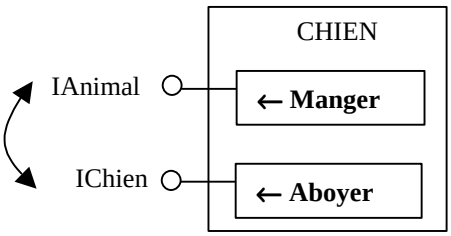
Le module de classe se comporte comme un serveur qui attend qu'un client lui demande de retourner une propriété ou exécuter une méthode. Le module client instantie la classe et appelle les méthodes et propriétés :



4.1.3. Query Interfaces (QI)

Lorsqu'une classe donnée possède plusieurs interfaces et qu'on dispose d'une variable pointant sur une des interfaces, il faut faire une "**Query Interface**" pour récupérer un pointeur sur une autre interface, afin de pouvoir utiliser les méthodes et propriétés de cette deuxième interface. Par exemple, avec l'exemple précédent, on déclare une deuxième variable pointant sur la deuxième interface puis on récupère le pointeur qui pointait sur l'ancien objet.

```
'Création d'un nouveau Chien avec
l'interface IChien
Dim pChien as IChien
Set pChien = New Chien
pChien.Aboyer

'QueryInterface pour utiliser la méthode
Manger
Dim pAnimal as IAnimal
Set pAnimal = pChien
pAnimal.Manger
'pChien et pAnimal pointent sur le même
objet
```



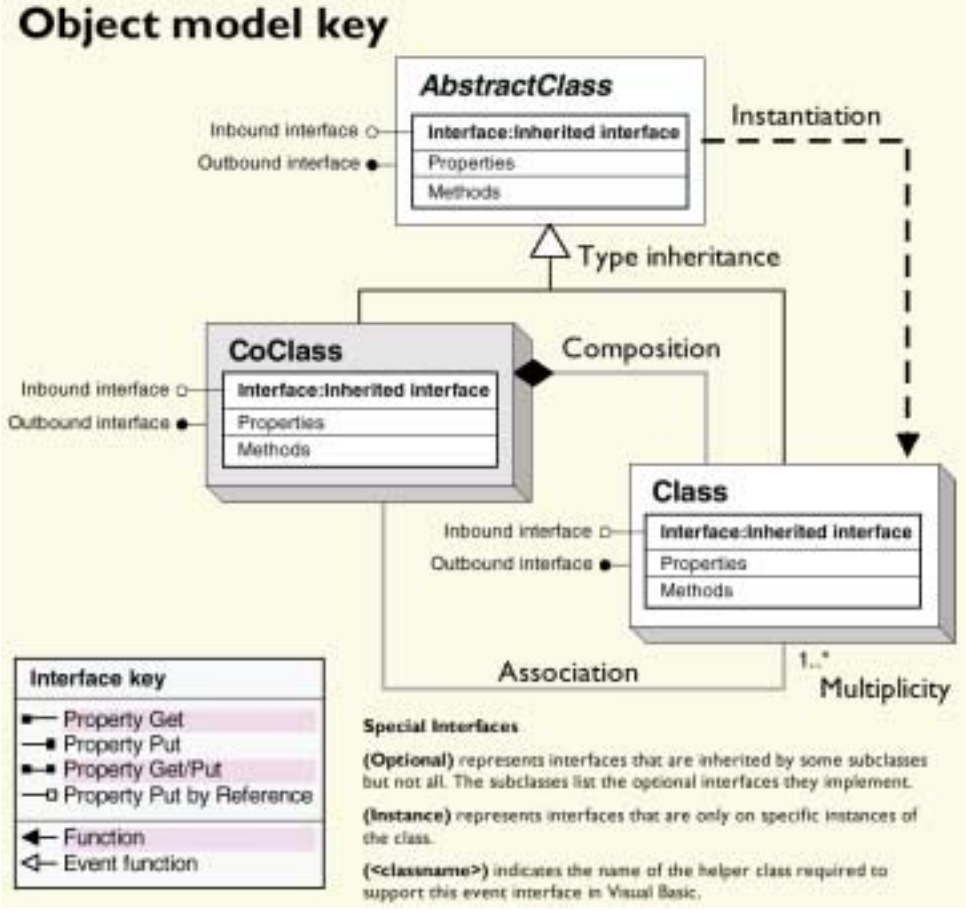
Ce mécanisme de Query Interface est très fréquent lorsqu'on développe avec ArcObjects.

4.2. OBJECT MODEL DIAGRAMS

Les ArcObjetcs OMD utilisent les notations UML (Unified Modeling Language) avec quelques modifications pour tenir compte des constructions spécifiques à la norme COM. Ils sont disponibles en format pdf dans l'aide en ligne "ArcObjects Developer Help" :

- ArcGIS : version simplifiée de l'ensemble des diagrammes
- All Object Model Diagrams (tous les diagrammes détaillés dans un seul fichier, pour faciliter les recherches de chaînes de caractères).
- Application Framework
- ArcCatalog
- ArcMap
- ArcMap Editor
- ArcObjects Controls
- Display
- Geocoding
- Geodatabase
- Geodatabase (Supplemental)
- Geometry
- IMS
- Labeling and Annotation
- Map Layer
- Network
- Output
- Raster
- Spatial Reference
- Styles
- TIN
- 3D Analyst Extension
- Spatial Analyst Extension
- StreetMap USA Extension

Légende des diagrammes :

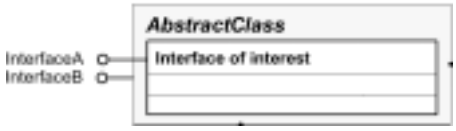


4.2.1. Classes

On distingue 3 types de classes dans les diagrammes UML : les abstract classes, les coclasses, et les classes :

4.2.1.1. Abstract Class

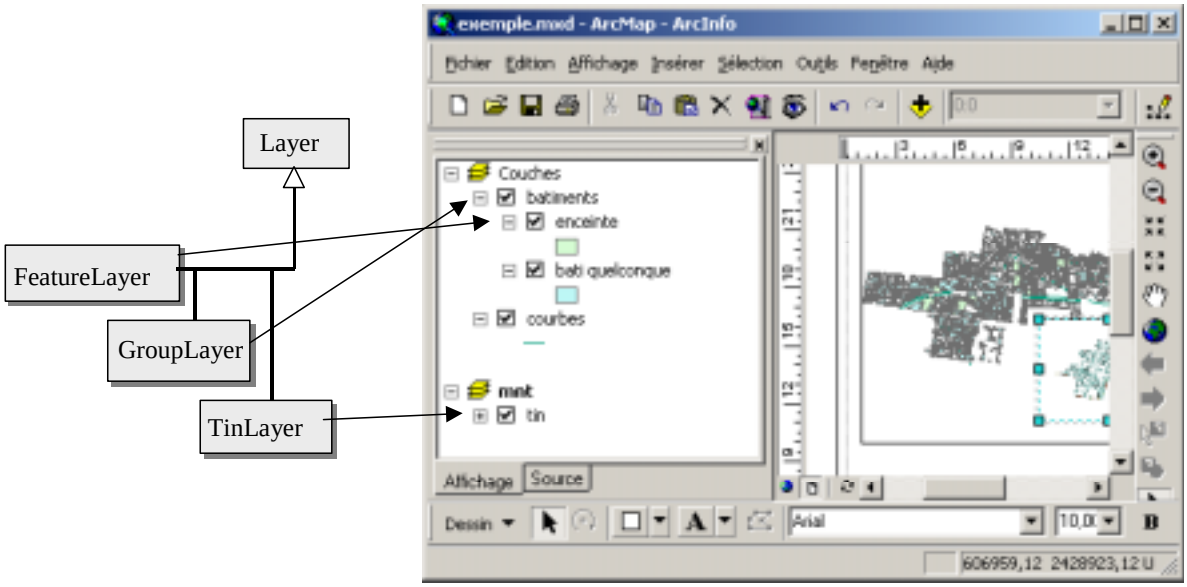
Une Abstract Class est représentée sur les OMD par un rectangle 2D légèrement grisé :



Elle définit des propriétés générales communes à plusieurs "sous-classes". **On ne peut pas créer** (instantier) d'objets avec ce type de classe.

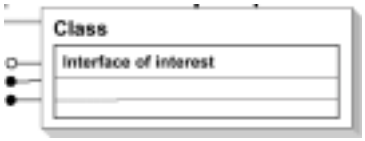
Par exemple, dans le "MapLayer OMD", on dispose de l'abstract class "Layer", correspondant à une couche dans ArcMap. L'abstract class "Layer" définit des

propriétés générales telles que le nom ou les références spatiales de la couche. Les propriétés spécifiques à chaque type de couche sont portées par les "sous-classes" (FeatureLayer = couches de données vecteur, GroupLayer = groupe de couches, TinLayer = Triangular Irregular Network)



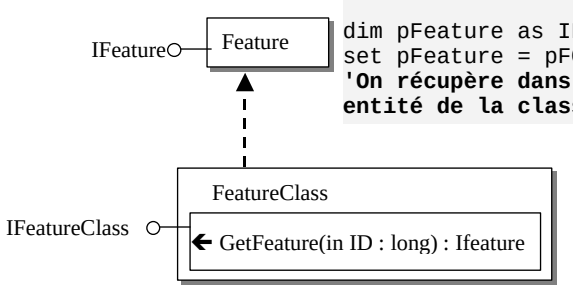
4.2.1.2. Class

Une "class" est représentée sur les OMD par un rectangle 3D blanc :



Une "class" est dite "instantiable" mais pas "creatable": elle ne peut pas créer directement un nouvel objet. Un objet d'une "class" peut être créé comme propriété d'une autre classe ou grâce à une méthode d'une autre classe.

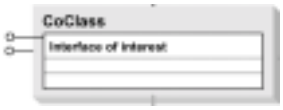
Par exemple, dans le "GeoDatabase OMD", on dispose de la "class" "FeatureClass" correspondant à une classe d'entité ArcGIS. Un objet de cette classe permet d'instancier un objet de la classe "Feature" (une entité) grâce à la méthode "GetFeature" par exemple.




```
dim pFClass as IFeatureClass
set pFClass = ...'initialisation de pFClass

dim pFeature as IFeature
set pFeature = pFClass.GetFeature(i)
'On récupère dans pFeature un pointeur sur la ième
entité de la classe d'entité pointée par pFClass
```

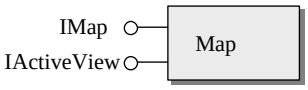
4.2.1.3. CoClass

Une "class" est représentée sur les OMD par un rectangle 3D grisé :



Une "CoClass" (Component Object Class) est dite "creatable". Un objet d'une CoClass peut être directement créé à l'aide du mot-clé VBA "New".

Par exemple, dans "ArcMap OMD", on dispose de la CoClass "Map", correspondant dans ArcMap à un bloc de données.



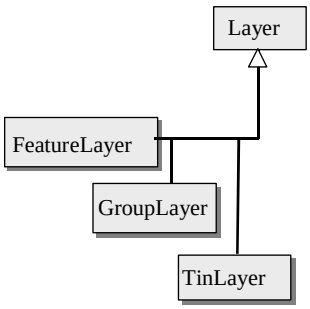

```
dim pMap as IMap
set pMap = New Map
'Création d'un nouveau bloc de données
```

4.2.2. Relations

Différents types de relations sont possibles entre ces différentes classes :

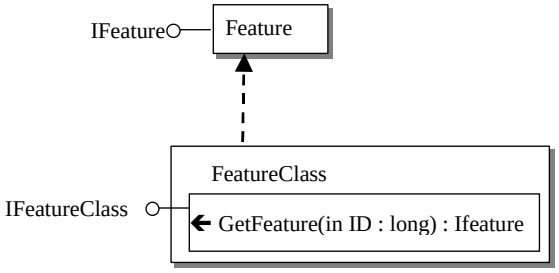
4.2.2.1. Héritage de type

L'héritage définit une relation entre une classe et une sous-classe. Les propriétés et méthodes de la "super-classe" sont partagées par l'ensemble de ses sous-classes. Dans l'exemple ci-dessous, les sous-classes (CoClass) FeatureLayer, GroupLayer et TinLayer héritent des propriétés et méthodes de l'abstract classe Layer.



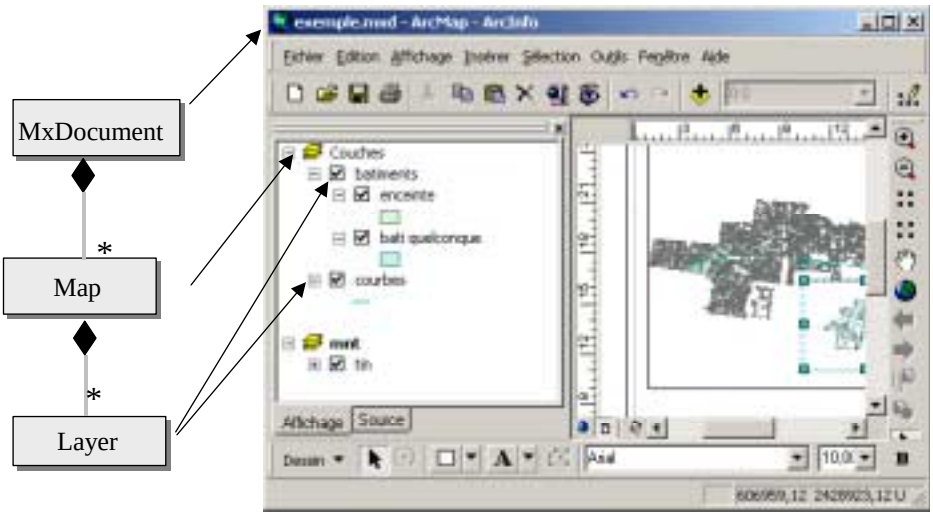
4.2.2.2. Instantiation

Une relation d'instanciation spécifie qu'un objet d'une classe possède une méthode pouvant créer un objet d'une autre classe, ce qui correspond à l'exemple précédent entre les classes "FeatureClass" et "Feature".



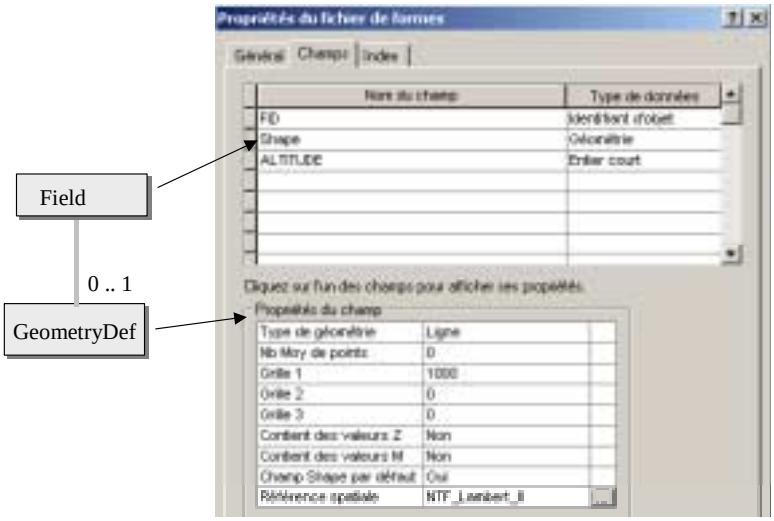
4.2.2.3. Composition

Dans une relation de composition, les objets de la classe "parent" contrôlent l'existence des objets de la classe "enfants".
Dans l'exemple ci-dessous, si on détruit un objet "Map" (donc un bloc de données ArcMap), on détruit également les "Layers" composant cette "Map" (donc les couches du bloc de données) :



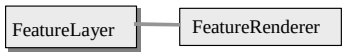
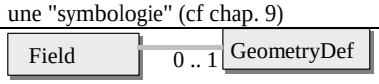
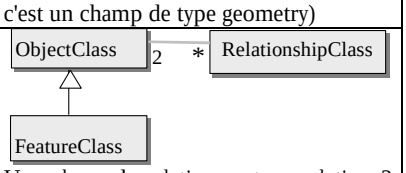
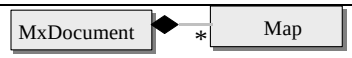
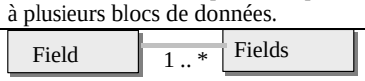
4.2.2.4. Association

Le dernier type de relation est la relation d'association, qui signifie juste que deux classes sont associées entre elles. Par exemple, dans le "geodatabase OMD", on voit qu'un objet "Field" (un attribut) peut être associé à un "GeometryDef" (objet qui définit des propriétés géométriques), ce qui est le cas de l'attribut "Shape" :



4.2.2.5. Cardinalités des relations

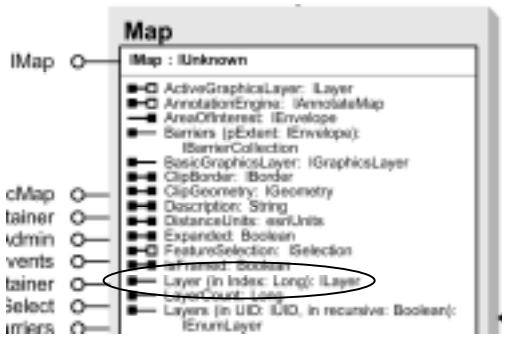
Pour les relations de composition ou d'association, la cardinalité est indiquée de chaque côté de la relation et renseigne sur le nombre d'objets pouvant être associés avec un objet donné. Les notations sont les suivantes :

1	Un objet ne peut être en relation qu'avec un et un seul objet. C'est l'option par défaut.	 Une couche vecteur est en relation avec une "symbologie" (cf chap. 9)
0..1	Un objet peut être en relation avec zéro ou un objet	 Un objet "Field" peut éventuellement être être associé à un "GeometryDef" (si c'est un champ de type geometry)
M..N	M objets en relation avec N objets	 Une classe de relation met en relation 2 tables.
***** ou **0..***	Un objet peut être en relation avec 0 à plusieurs objets	 Un document ArcMap est composé de 0 à plusieurs blocs de données.
1..*	Un objet peut être en relation avec 1 ou plusieurs objets	 Un "Fields" (collection de champs correspondant à l'ensemble des champs d'une table) est associé à 1 ou plusieurs champs.

4.2.3. Propriétés

4.2.3.1. Propriété en lecture seule

Notation : 



L'utilisation est différente selon qu'elle retourne une valeur ou bien une interface :
- Valeur (String, Double etc ...)

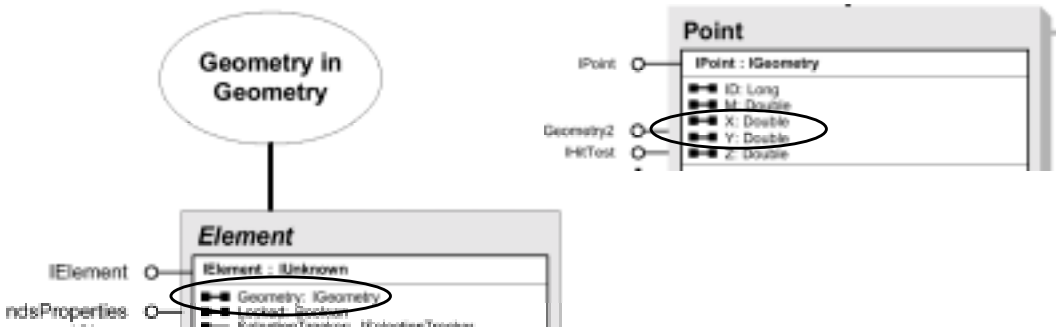
```
Dim icount as integer
icount = pMap.LayerCount
```

- Interface (utilisation du mot-clé "Set")

```
Dim pFLayer as iIFeatureLayer
Set pFlayer = pMap.Layer(0)
```

4.2.3.2. Propriété en lecture/"écriture par valeur"

Notation : 

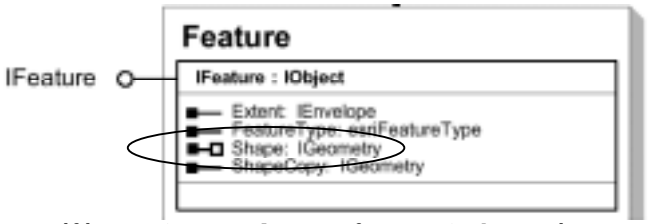


Exemple d'utilisation en écriture par valeur (set ByVal)

```
Dim pElement as IElement
Set pElement = New Element 'Création d'un Elément graphique
Dim pPoint as Ipoint
Set pPoint = New Point 'Création d'un Point
pPoint.X = 5    'On affecte au Point les coordonnées
pPoint.Y = 10   'x = 5, y = 10
pElement.Geometry = pPoint 'On affecte le Point à la géométrie de l'élément
```

4.2.3.3. Propriété en lecture/"écriture par réference"

Notation : 



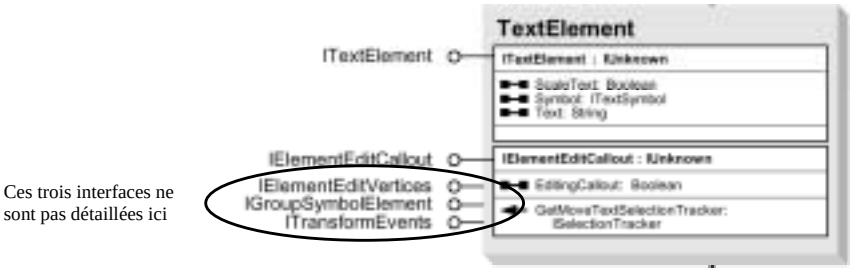
En mode écriture par référence (ie par adresse, cf. § 3.2.6), il est nécessaire d'utiliser le mot-clé "Set. ". Exemple d'utilisation en écriture par référence (set ByRef) :

```
Dim pFeature as IFeature
Set pFeature.shape = pPoint 'On affecte pPoint à la géométrie de l'entité pFeature
```

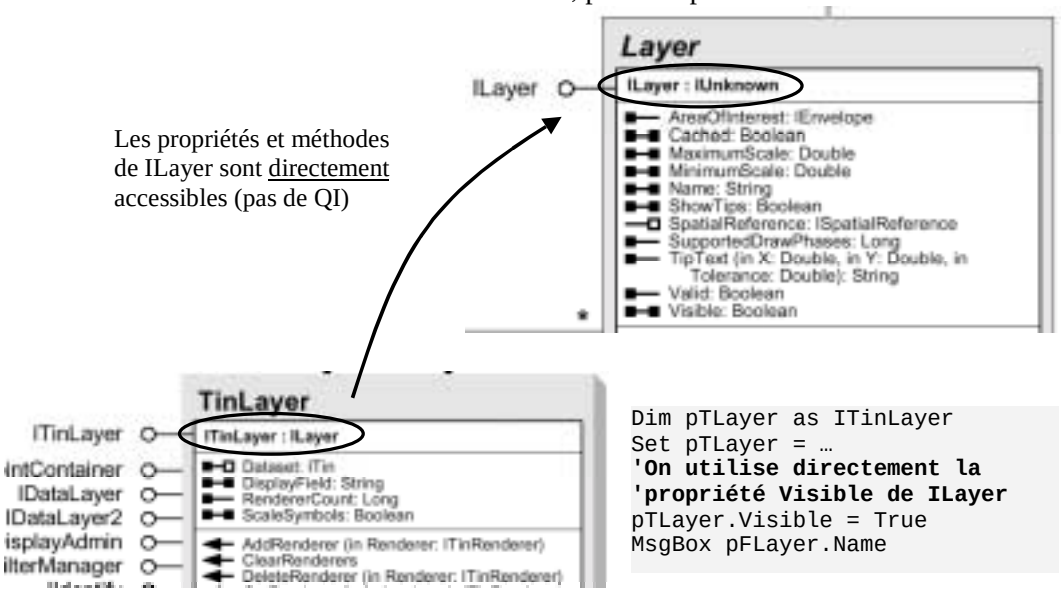
4.2.4. Interfaces

Toutes les interfaces ne sont pas décrites directement au niveau de la classe qui les implémente. Elles peuvent être décrites ailleurs dans le diagramme (généralement en bas à droite, indépendamment d'une classe), ou bien seulement dans l'aide en ligne :

Notation : **I ....** ○—



Dans certains cas, une interface hérite d'une autre interface (indiqué par la syntaxe **"Iinterface1 : Iinterface2"** en tête de description). Dans ce cas, toutes les méthodes et propriétés de l'interface parent sont utilisables directement sans Query Interface sur l'interface héritante. C'est l'**Interface Inheritance**, par exemple :




```
Dim pTLayer as ITinLayer
Set pTLayer = …
'On utilise directement la
'propriété Visible de ILayer
pTLayer.Visible = True
MsgBox pFLayer.Name
```

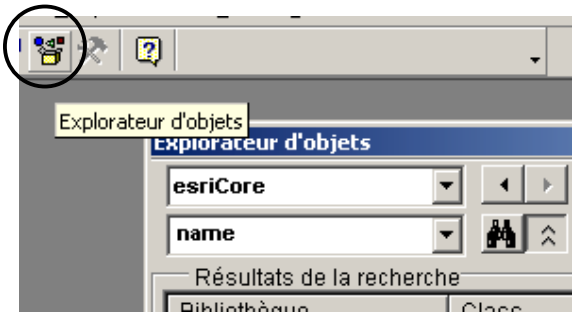
4.3. OU TROUVER DE L'AIDE ?

4.3.1. ArcObjects developer help

C'est l'aide en ligne d'ArcObjects (Démarrer/Programmes/ARCGIS/ArcObjects developer help). Outre les diagrammes (OMD), elle contient des renseignements détaillés sur les classes, interfaces ... ainsi que des exemples et des documents techniques.

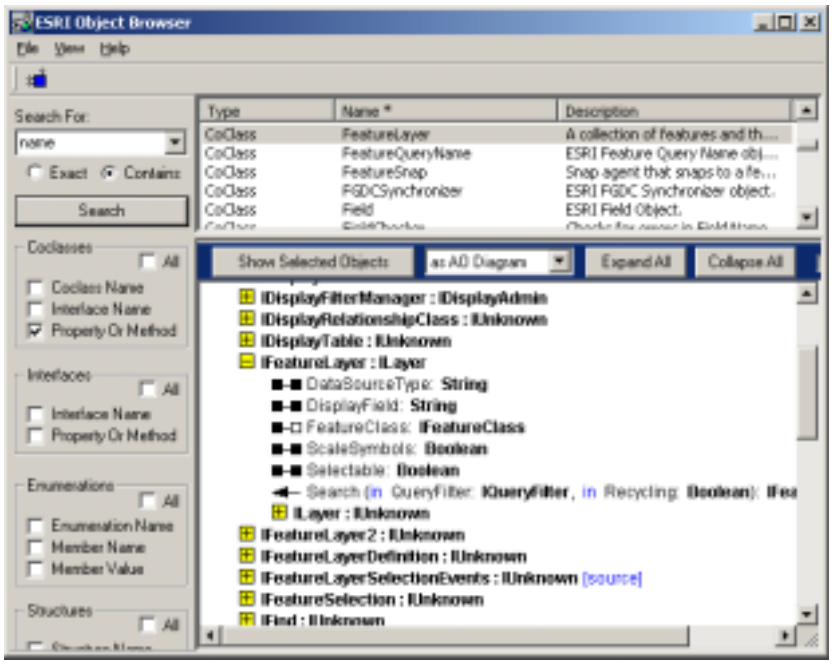
4.3.2. Explorateur d'objets (Editeur VBA)

C'est l'explorateur standard de l'éditeur VBA. Il affiche les classes, propriétés, méthodes, événements et constantes disponibles dans les bibliothèques d'objets et les procédures du projet. Il permet de rechercher et d'utiliser des objets créés par le développeur ainsi que des objets provenant d'autres applications.
La bibliothèque d'ArcObjects est la bibliothèque **"EsriCore" :**



4.3.3. Esri Object Browser

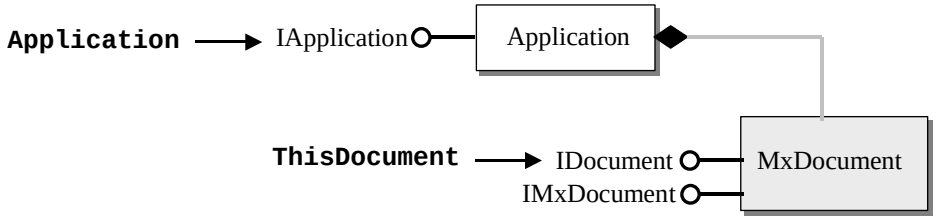
Il permet d'explorer la structure d'ArcObjects de manière plus précise qu'avec l'explorateur d'objets VBA. Plusieurs types de visualisation sont disponibles : as VB, as IDL, as AO Diagram (mêmes notations que les diagrammes)



L'application "Esri Object Browser" est stockée dans le répertoire :
"< repertoire d'installation d'ArcGIS>\arcexe81\ArcObjects Developer Kit\Utilities**EOBrowser.exe**"

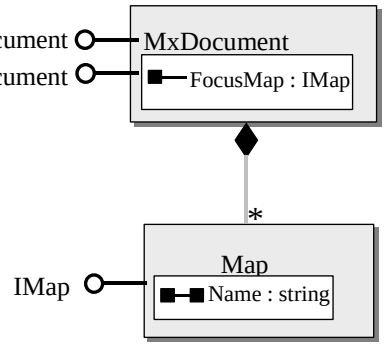
4.4. GLOBAL VARIABLES SCOPE

ArcObjects fournit deux variables prédéfinies disponibles à n'importe quel moment dans le code et qui sont les deux points d'entrée dans le modèle. Ces deux variables "Application" et "ThisDocument" pointent respectivement sur l'interface IApplication de la classe Application et sur l'interface IDocument de la classe MxDocument (un MxDocument correspond à un document ArcMap).



Ces deux variables n'ont pas besoin d'être déclarées ni initialisées, elles sont directement utilisables. Par exemple, le code suivant permet d'afficher le nom de la carte courante :

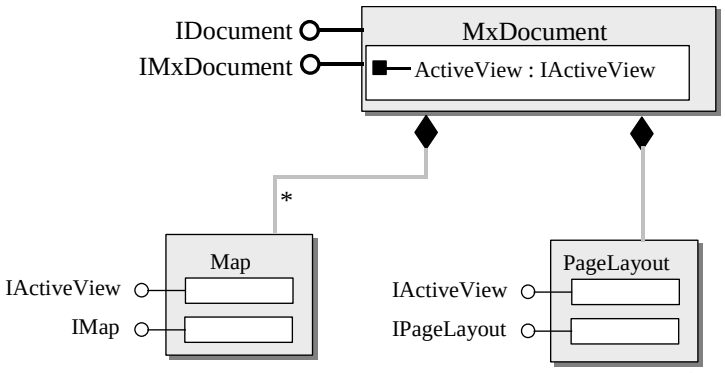
```
Dim pMxDoc as IMxDocument

'Query Interface pour travailler sur l'interface
IMxDocument
set pMxDoc = ThisDocument

dim pMap as IMap
'On récupère dans pMap un pointeur sur la carte
active (focus map)
set pMap = pMxDoc.FocusMap

'On affiche le nom de la carte dans un msgbox
MsgBox pMap.Name

Le code suivant est équivalent :
Dim pMxDoc as IMxDocument
set pMxDoc = ThisDocument
MsgBox pMxDoc.FocusMap.Name
```



4.5. EXEMPLES : MANIPULATION D'OBJETS ARCMAP

4.5.1. ActiveView

Un document ArcMap possède toujours une vue active ("ActiveView") qui est soit le "mode données" (map), soit le "mode mise en page" (layout), l'utilisateur pouvant passer de l'un à l'autre. Pour manipuler la notion d'ActiveView, il faut passer par l'interface IActiveView supportée par deux classes Map et PageLayout qui implémentent l'interface de deux manières différentes (polymorphisme) :



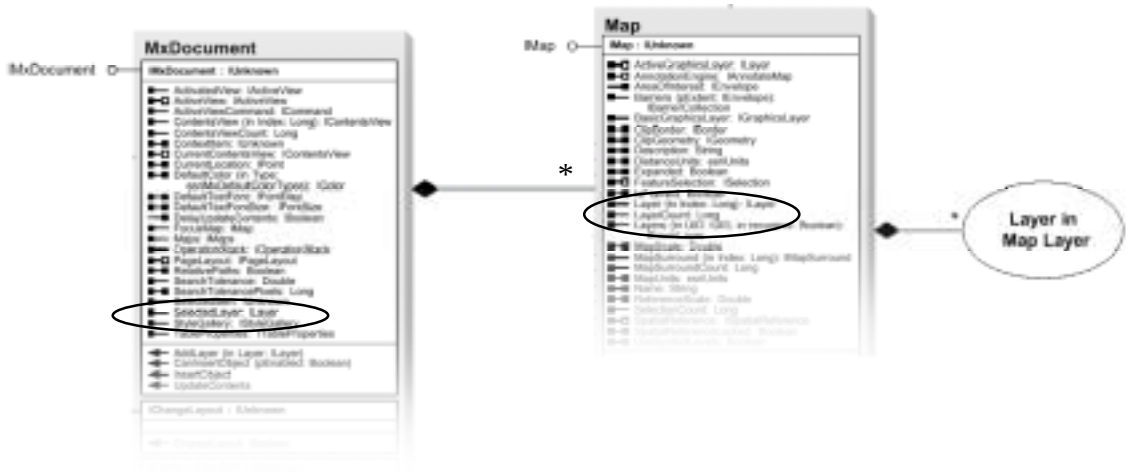
(ArcMap OMD)

Comment tester le type d'ActiveView ?
Le code VBA suivant teste le type de vue active, grâce au mot-clé "TypeOf" :

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument 'QI entre IMxDocument et IDocument
If TypeOf pMxDoc.ActiveView Is IMap Then
  MsgBox "La vue active est le mode données"
ElseIf TypeOf pMxDoc.ActiveView Is IPageLayout Then
  MsgBox "La vue active est le mode mise en page"
End If
```

4.5.2. Layers

Un document (MxDocument) est composé de zéro à plusieurs bloc de données (Map), eux-mêmes composés de zéro à plusieurs couches (Layer). Pour accéder à une couche, on peut passer soit par le document (propriété SelectedLayer), soit par le bloc de données (propriétés Layer et LayerCount) :



(ArcMap OMD)

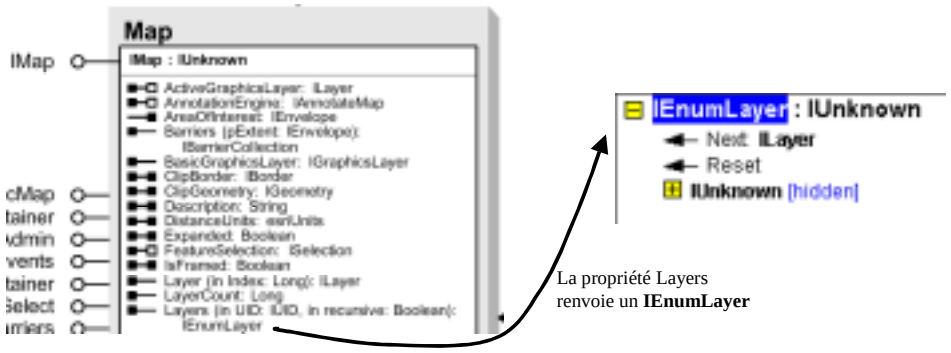
Comment accéder à une couche via le document ?
Le code VBA suivant récupère dans un pointeur la couche sélectionnée. Pour l'utiliser, il faut au préalable vérifier que le pointeur ne pointe pas sur rien (mot-clé "Nothing"), c'est à dire qu'une couche est effectivement sélectionnée par l'utilisateur d'ArcMap :

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Dim pLayer As ILayer
'On initialise pLayer avec la couche sélectionnée
Set pLayer = pMxDoc.SelectedLayer
If pLayer Is Nothing Then
  'Si pLayer pointe sur rien, on affiche un message
  MsgBox "Vous devez sélectionner une couche !"
Else
  'sinon on affiche le nom de la couche
  MsgBox pLayer.Name
End If
```

Comment accéder à une couche par son nom ?
On ne peut pas accéder directement à une couche par son nom. On doit passer par le bloc de données contenant la couche et parcourir l'ensemble des couches du bloc de données avec une boucle en testant le nom. La fonction suivante retourne la couche correspondant au bloc de données et au nom passé en entrée de la fonction :

```
Function FindLayerByName(pMap as IMap, sName as String) As ILayer
  Dim i as Integer
  'Boucle sur le nombre de couche (numérotées à partir de 0)
  For i = 0 to pMap.LayerCount – 1
    'Si la ième couche porte le nom recherché, la fonction
    ' retourne cette couche
    If pMap.Layer(i).Name = sName Then
      Set FindLayerByName = pMap.Layer(i)
    End If
  Next
End Function
```

Utilisation d'une énumération pour parcourir l'ensemble des couches :
Au lieu d'utiliser une boucle "For ...Next" pour parcourir l'ensemble des couches, il est possible d'utiliser le mécanisme d'énumérateur. Un énumérateur est similaire à un tableau à une dimension avec seulement deux méthodes pour parcourir l'énumérateur : Reset et Next. Il va s'utiliser généralement avec des boucles "Do ... While" ou "Do ... Until". Par exemple :



Le code VBA suivant affiche dans un MsgBox le nombre de couches d'un bloc de données ainsi que leurs noms :

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pMap As IMap
Set pMap = pMxDoc.FocusMap
Dim strNameList As String
strNameList = "La carte contient " & pMap.LayerCount _
              & "couches :" & vbCrLf
Dim pEnumLayer As IEnumLayer
'On récupère un énumérateur sur les couches du bloc de données
Set pEnumLayer = pMap.Layers
pEnumLayer.Reset
Dim pLayer As ILayer
'On se place sur la première couche de l'énumérateur
Set pLayer = pEnumLayer.Next
Do While Not pLayer Is Nothing
  'Tant que l'énumérateur n'a pas été entièrement parcouru,
  'on ajoute le nom de la couche à strNameList
  strNameList = strNameList & pLayer.Name & vbCrLf
  ' et on passe à la couche suivante de l'énumérateur
  Set pLayer = pEnumLayer.Next
Loop
'On affiche strNameList
MsgBox strNameList
```

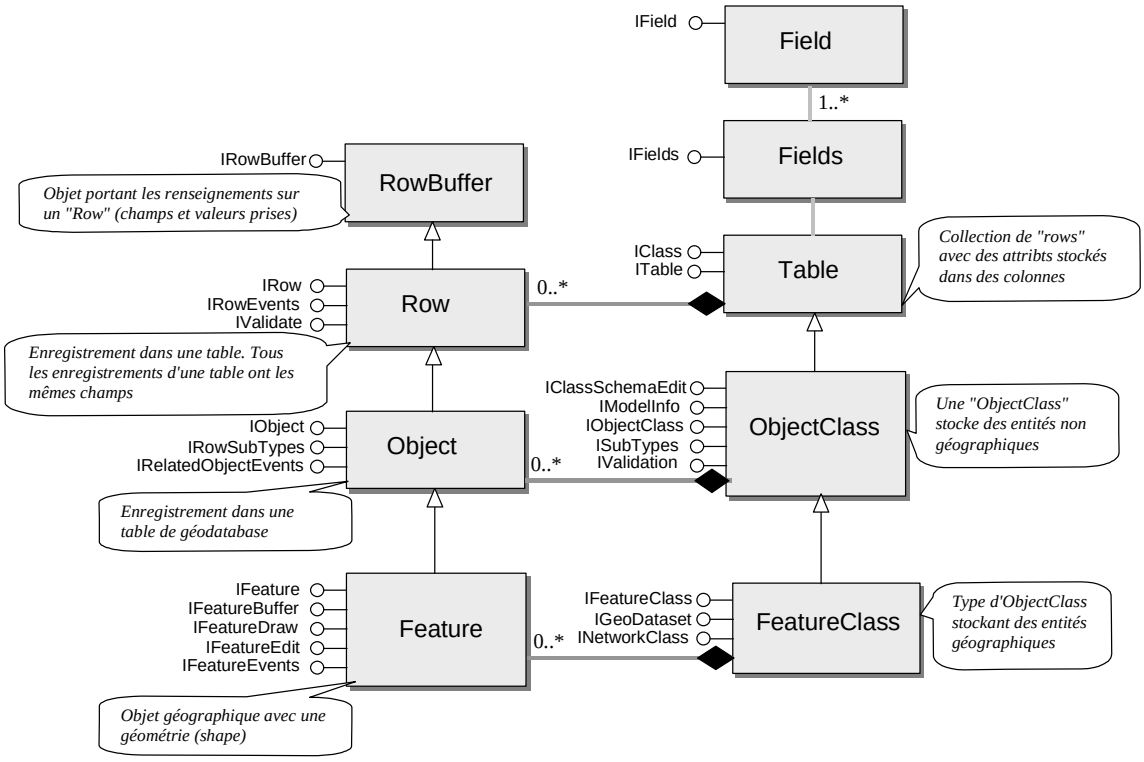
(remarque : noter l'utilisation du caractère "_" pour couper une ligne de code trop longue)

5. Lecture de champs, requête sémantique, travail sur la sélection

5.1. FEATURECLASS ET FEATURE

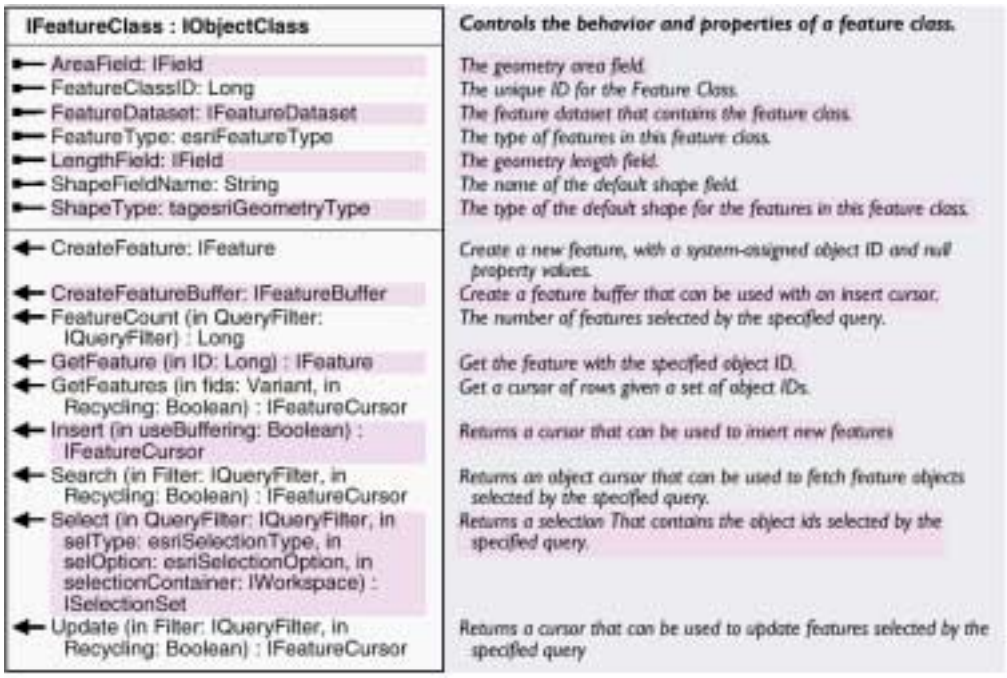
5.1.1. Présentation générale

Le schéma suivant présente les différentes classes nécessaires à la manipulation d'entités (extrait du Geodatabase OMD) :

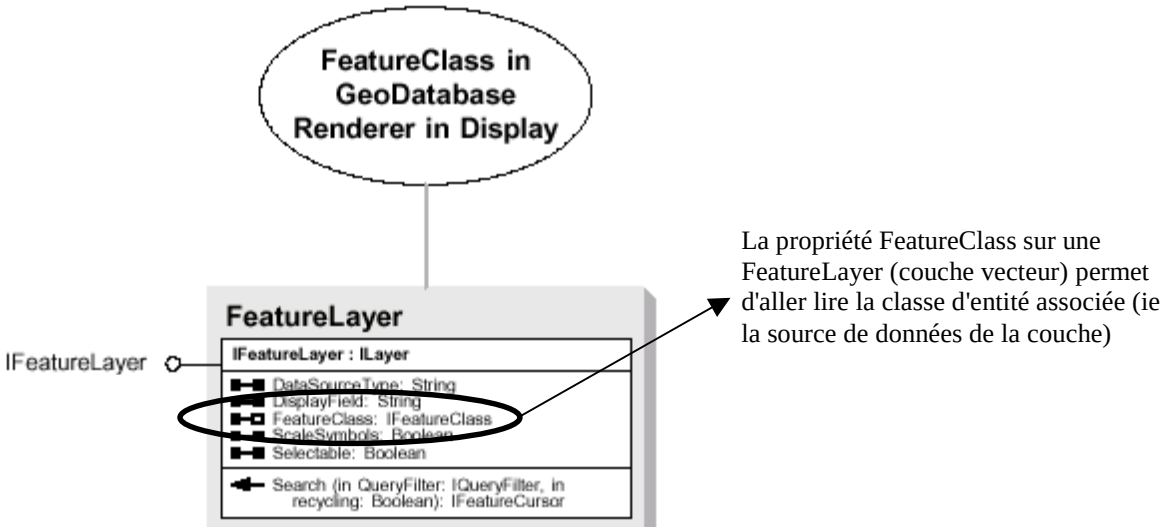


Une "FeatureClass" (classe d'entité au sens large, pas seulement de géodatabase) est une "ObjectClass" dont les objets sont des entités (Feature). Toutes les Feature d'une FeatureClass ont les mêmes attributs.

5.1.2. IFeatureClass



Comment accéder à une FeatureClass via un document ArcMap ?



(Map Layer OMD)

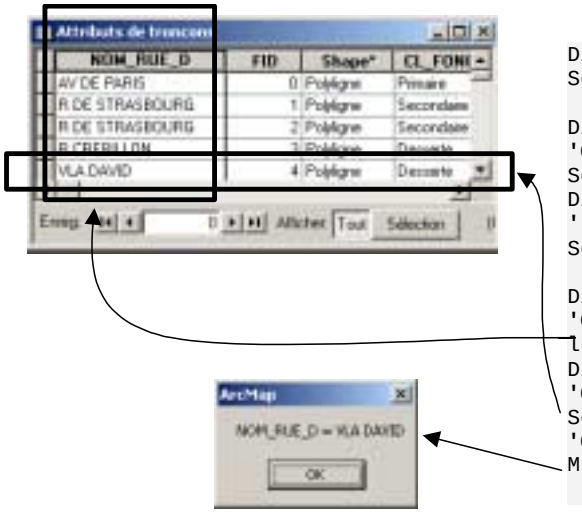
Le code VBA suivant récupère dans un pointeur la FeatureClass correspondant à la première couche de la carte active :

```
Dim pMxDoc as IMxDocument
Set pMxDoc = ThisDocument
Dim pFLayer as IFeatureLayer
'On récupère la première couche de la focus map
Set pFLayer = pMxDoc.FocusMap.Layer(0)
Dim pFClass as IFeatureClass
'On lit la classe d'entité associée à cette couche
Set pFClass = pFLayer.FeatureClass
```

5.1.3. Lecture de valeurs de champs

La méthode "GetFeature (in ID : Long)" sur IFeatureClass permet de récupérer l'entité (Feature) dont l'OID est celui passé en entrée. La propriété "Value (in Index : Long)" sur IRowBuffer retourne la valeur prise par cette entité pour le champ de rang "Index" passé en entrée. On peut directement utiliser la propriété Value sur IFeature grâce aux **héritage d'interface** (cf. 4.2.4) successifs.

Comment lire les valeurs de champs dans une table ?
Le code VBA ci-dessous affiche dans un MsgBox la valeur de l'attribut "NOM_RUE_D" de la classe d'entité associée à la 3ème couche du bloc de données actif.




```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Dim pFLayer As IFeatureLayer
'On récupère la 3ème couche de la carte
Set pFLayer = pMxDoc.FocusMap.Layer(2)
Dim pFClass As IFeatureClass
' .. et la classe d'entité associée
Set pFClass = pFLayer.FeatureClass

Dim lField As Long
'On recherche le champ "NOM_RUE_D"
lField = pFClass.FindField("NOM_RUE_D")
Dim pFeature As IFeature
'On recherche l'entité dont FID = 4
Set pFeature = pFClass.GetFeature(4)
'On affiche la valeur du champ pour cette entité
MsgBox "NOM_RUE_D = " & pFeature.Value(lField)
```

5.2. ENUMERATION DES OBJETS SELECTIONNES SUR LA CARTE

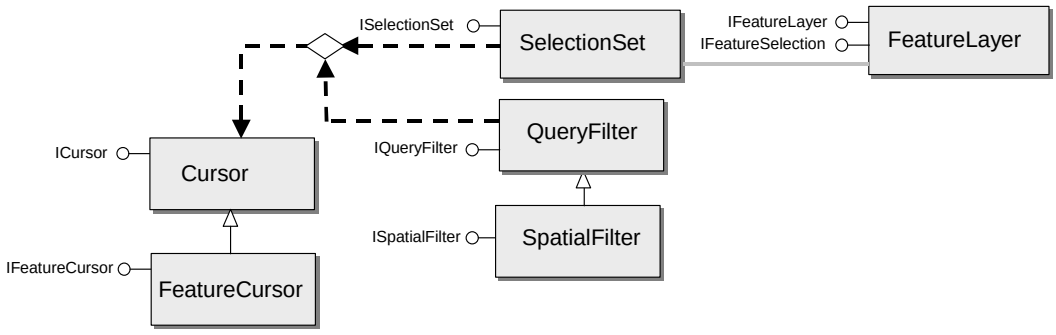
Une des méthodes pour parcourir la sélection est d'utiliser un "énumérateur" (similaire à un tableau à une dimension avec deux méthodes : Next et Reset pour parcourir son contenu).

```
Dim pEnumFeat As IEnumFeature
Dim pFeat As IFeature
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
'QueryInterface entre ISelection et IEnumFeature
Set pEnumFeat = pMxDoc.FocusMap.FeatureSelection
'On se place sur le premier élément de l'énumérateur
Set pFeat = pEnumFeat.Next
Dim i As Integer
Do While (Not pFeat Is Nothing)
  'on incrément i à chaque passage dans la boucle
  i = i + 1
  Set pFeat = pEnumFeat.Next
Loop
MsgBox i & " objets sélectionnés !"
```

NB : La propriété "SelectionCount sur l'interface IMap renseigne également sur le nombre d'objets sélectionnés.

5.3. CURSEUR SUR LES OBJETS SELECTIONNES (SUR UNE COUCHE)

On peut également parcourir la sélection grâce au mécanisme de "curseur" (cursor). Un curseur est un pointeur qui donne accès à un enregistrement à la fois. Les curseurs permettent entre autre de parcourir un ensemble d'enregistrements sélectionnés sans perdre la sélection. Ils sont par exemple très utiles pour lire ou écrire une valeur d'attribut. Un "FeatureCursor" peut être instancié par un "SelectionSet" (avec la méthode "Search") qui représente l'ensemble des objets sélectionnés pour une couche donnée (accessible via la propriété "SelectionSet" de IFeatureSelection sur la classe FeatureLayer). Il est possible d'appliquer un filtre au SelectionSet pour restreindre les nombre d'enregistrements pris en compte par le curseur. Un "QueryFilter" filtre les objets selon un critère sémantique (requête SQL) alors que le "SpatialFilter" fait intervenir des critères spatiaux. Le diagramme est le suivant (extrait de Geodatabase OMD) :



Le code VBA suivant affiche dans un MsgBox la valeur de l'attribut "NOM_RUE_D" pour les entités de la couche n°2 sélectionnées :

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pFLayer As IFeatureLayer
Set pFLayer = pMxDoc.FocusMap.Layer(2)
Dim pFeature As IFeature
Dim pFeatureSelection As IFeatureSelection
Dim pSelectionSet As ISelectionSet
Dim pFeatureCursor As IFeatureCursor
'On fait une QI entre IFeatureLayer et IFeatureSelection
Set pFeatureSelection = pFLayer
Set pSelectionSet = pFeatureSelection.SelectionSet
'Création du curseur avec la méthode "Search"
pSelectionSet.Search Nothing, False, pFeatureCursor
'On pointe sur le premier élément du curseur avec la propriété Next
Set pFeature = pFeatureCursor.NextFeature
' boucle "tant qu'on point sur quelque chose"
Do While Not pFeature Is Nothing
  MsgBox "NOM_RUE_D = " & pFeature.Value(pFeature.Class.FindField("NOM_RUE_D"))
  'On passe à l'élément suivant
  Set pFeature = pFeatureCursor.NextFeature
Loop
```

La syntaxe de la méthode "Search" sur ISelectionSet est la suivante :

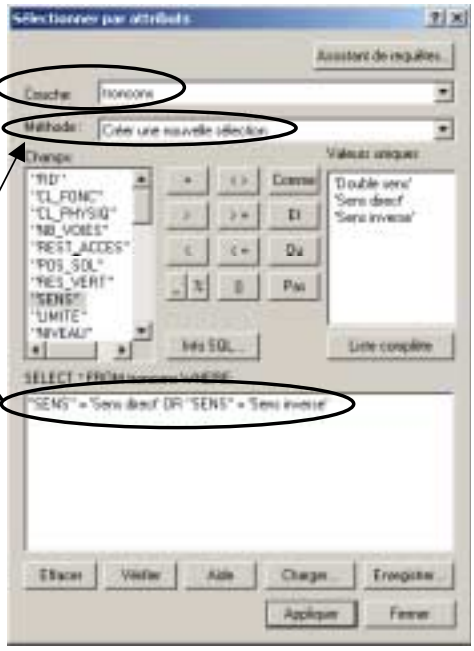
```
object.Search (In pQueryFilter : IQueryFilter, in Recycling : Boolean, out
ppCursor : ICursor)
```

- Si on n'utilise pas de filtre (IQueryFilter), on passe "Nothing"
- Si le paramètre "**Recycling**" prend la valeur True, la même adresse mémoire est utilisée pour chaque "NextFeature", d'où des performances meilleures (à utiliser en mode lecture seule). Si le paramètre prend la valeur "False", un nouvel emplacement mémoire est alloué pour chaque entité du curseur. Il est nécessaire d'utiliser cette valeur (nonrecycling cursor) lorsque l'on souhaite modifier l'objet pointé par le curseur.

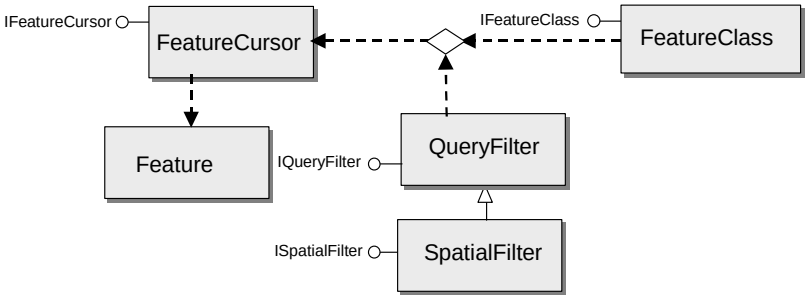
5.4. REQUETES SEMANTIQUE AVEC LES QUERY FILTERS

Pour sélectionner des entités selon un critère sémantique (de type requête SQL), on va utiliser les "QueryFilter". Ils sont utilisés par diverses méthodes : sélection sur une couche, création d'un curseur, dénombrement des entités d'une classe d'entité ... L'exemple suivant décrit l'utilisation d'un QueryFilter pour faire l'équivalent d' un "Sélectionner par attributs" sous ArcMap :

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pFLayer As IFeatureLayer
Set pFLayer = pMxDoc.FocusMap.Layer(2)
Dim pQueryFilter As IQueryFilter
'Création d'un nouveau QueryFilter
Set pQueryFilter = New QueryFilter
'La propriété WhereClause permet de spécifier le critère
pQueryFilter.WhereClause = "SENS = 'Sens direct'_
                        OR SENS = 'Sens inverse'"
Dim pFeatureSelection As IFeatureSelection
'QI entre IFeatureLayer et IFeatureSelection
Set pFeatureSelection = pFLayer
'Execution de la requête sur la couche pFeatureSelection
pFeatureSelection.SelectFeatures pQueryFilter _
                            , esriSelectionResultNew, False
pMxDoc.ActiveView.Refresh 'Rafraichissement de la vue
```



Un QueryFilter peut également être utilisé conjointement avec un FeatureCursor pour parcourir des entités extraites d'une FeatureClass suivant un critère sémantique (elles ne sont alors pas "sélectionnées à l'écran). Le schéma est le suivant :



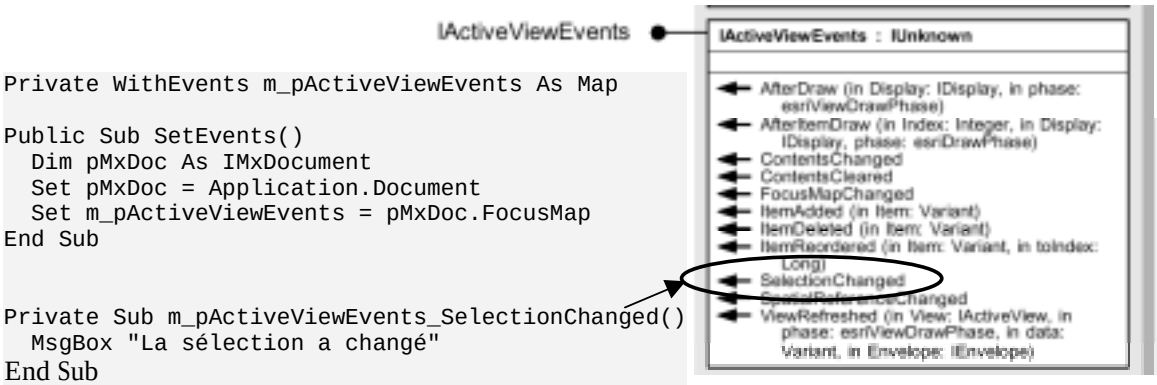
NB : Les méthodes "Add" et "Clear" sur IFeatureSelection permettent respectivement d'ajouter un élément à la sélection ou d'effacer la sélection.

5.5. RECUPERATION D'EVENEMENT SUR LA SELECTION

L'interface IActiveViewEvents permet d'accéder aux événements pouvant survenir lorsque l'état de la vue active change : modification de la sélection, suppression d'un élément de la mise en page, changement de FocusMap etc ... Plusieurs CoClass implémentent cette interface mais de manière différente.

le code VBA suivant montre l'utilisation de IActiveViewEvents pour récupérer l'événement comme quoi la sélection a été modifiée. Ce code doit être écrit dans le module "**ThisDocument**" :

```
Private WithEvents m_pActiveViewEvents As Map

Public Sub SetEvents()
  Dim pMxDoc As IMxDocument
  Set pMxDoc = Application.Document
  Set m_pActiveViewEvents = pMxDoc.FocusMap
End Sub


Private Sub m_pActiveViewEvents_SelectionChanged()
  MsgBox "La sélection a changé"
End Sub
```

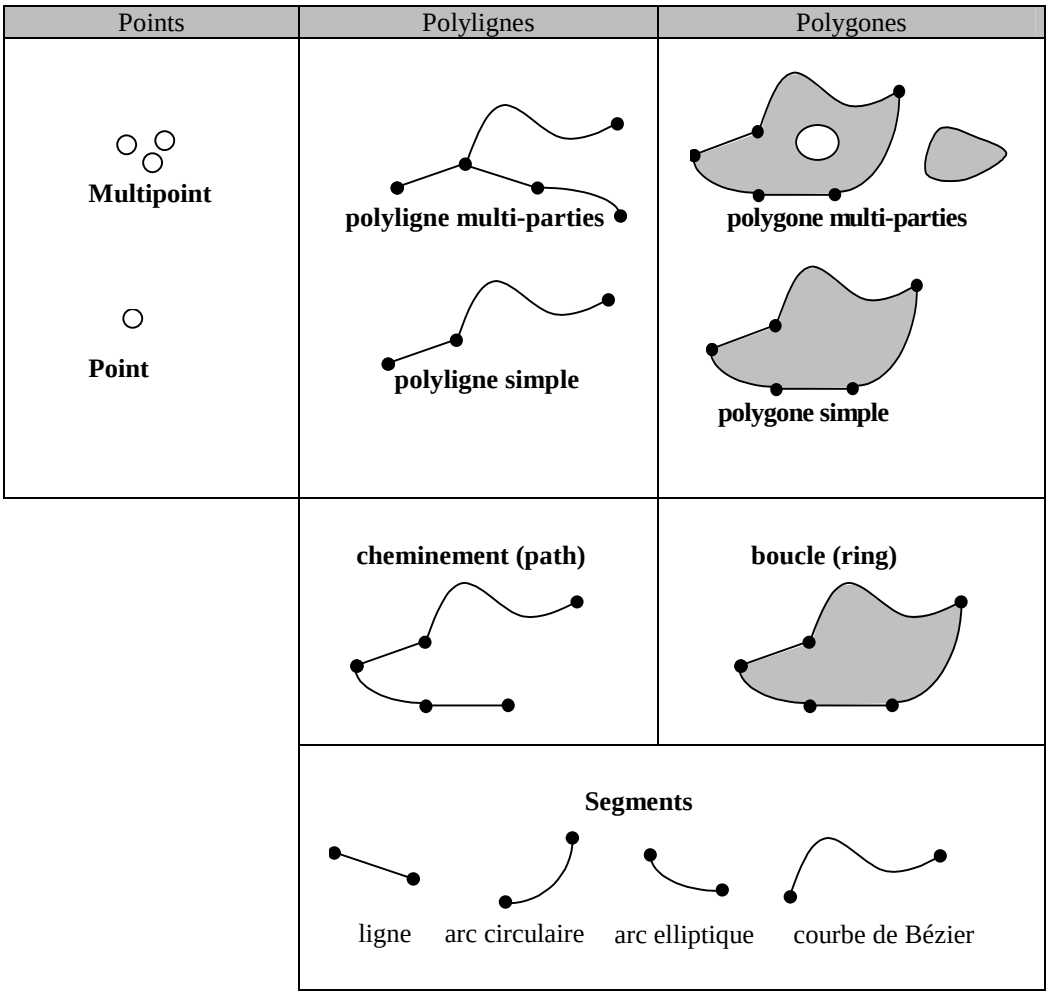


Extrait de la CoClass Map (ArcMap OMD)

Le mot-clé "**WithEvents**" indique que la variable est une variable objet utilisée pour répondre aux événements déclenchés par un objet ActiveX. Il est valide uniquement dans les modules de classe.

6. Géométrie

6.1. GEOMETRIE DANS UNE GEODATABASE



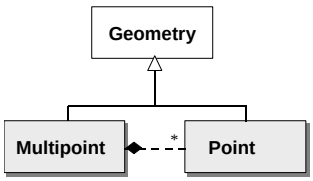
Remarques :
- pour le format shapefile, les segments sont seulement de type ligne.
- Les couvertures n'acceptent pas la multi-géométrie.

6.2. LA GEOMETRIE DANS ARCOBJECTS

Les objets décrits par le Geometry OMD permettent entre autres de localiser les données SIG. La localisation d'une entité (feature) est représentée par une instance de "Point", "Multipoint", "Polyline" ou "Polygon". Les "Envelopes", "Segments", "Rings" et "Paths" ne peuvent par contre pas représenter la localisation d'une entité mais sont utiles par exemple pour l'analyse spatiale, la représentation cartographique des données etc ...

6.2.1. Points et Multipoints

Un point est composé de deux coordonnées X et Y. Les multipoints sont des collections de points :



Par exemple, pour créer un nouveau point :

```
Dim pPoint As IPoint
Set pPoint = New Point
pPoint.X = 10
pPoint.Y = 20
'on peut également remplacer ces deux lignes par :
' pPoint.PutCoords 10,20
```

Puis un MultiPoint à partir de ce point :

```
Dim pMultiPts As IPointCollection
Set pMultiPts = New MultiPoint
MultiPts.AddPoint pPoint
```

6.2.2. Segments (Line, Circular Arc, BezierCurve)

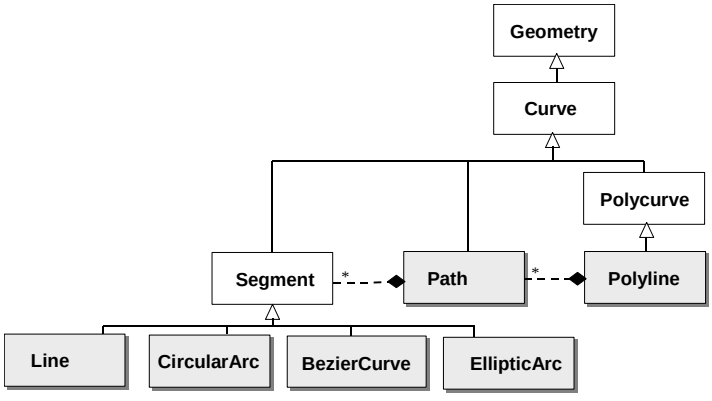
La classe "Segment" est une abstract classe avec quatre sous-classes : Line, Circular Arc, Bezier Curve et Elliptical Arc. La classe Segment est elle-même une sous-classe de "Curve". Les quatres classes Line, Circular Arc, Bezier Curve et Elliptical Arc héritent leurs principales propriétés et méthodes de Curve, entre autres les propriétés "FromPoint" et "ToPoint" qui définissent le début et la fin de n'importe quel type de segment.

Par exemple, pour construire un segment de type ligne, à partir de deux points pPointA et pPointB :

```
Dim pSegment as ILine
Set pSegment = New Line
pSegment.FromPoint = pPointA
pSegment.ToPoint = pPointB
```

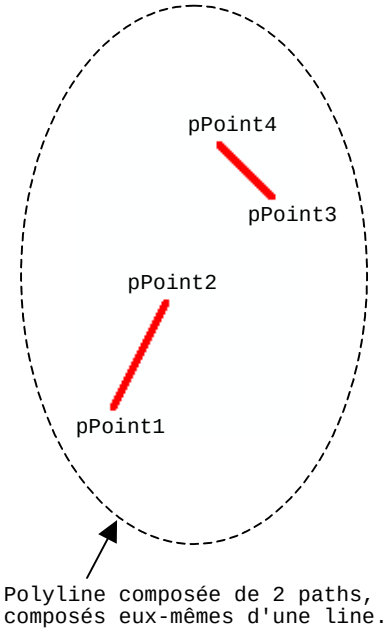
6.2.3. Paths et Polylines

Un path est la collection d'un ou plusieurs segments connectés. Une polyligne est une collection de paths connectés ou non.



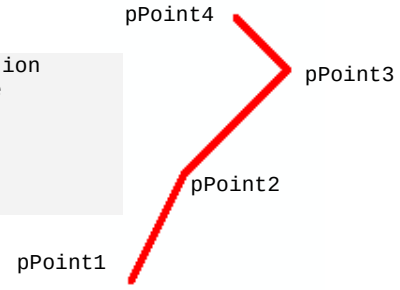
Par exemple, pour construire une polyligne composée de deux parties disjointes, on peut écrire le code VBA de la manière suivante :

```
Dim pPoint1 As IPoint, pPoint2 As IPoint
Dim pPoint3 As IPoint, pPoint4 As IPoint

Set pPoint1 = New Point
pPoint1.PutCoords 10, 10
Set pPoint2 = New Point
pPoint2.PutCoords 20, 30
Set pPoint3 = New Point
pPoint3.PutCoords 40, 50
Set pPoint4 = New Point
pPoint4.PutCoords 30, 60

Dim pGeometryColl As IGeometryCollection
Set pGeometryColl = New Polyline

Dim pSegmentColl As ISegmentCollection
Set pSegmentColl = New Path

Dim pLine As ILine
Set pLine = New Line
pLine.PutCoords pPoint1, pPoint2
pSegmentColl.AddSegment pLine
pGeometryColl.AddGeometry pSegmentColl

Set pSegmentColl = New Path
Set pLine = New Line
pLine.PutCoords pPoint3, pPoint4
pSegmentColl.AddSegment pLine
pGeometryColl.AddGeometry pSegmentColl
```



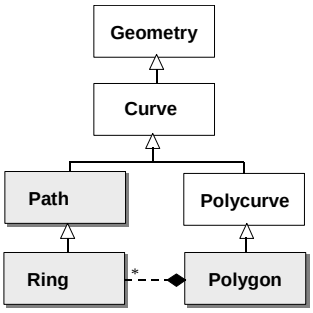
NB : pour construire une polyline "simple", c'est à dire composée d'une seule partie, il est plus simple d'utiliser la méthode "AddPoint" sur l'interface IPointCollection, implémentée entre autre par la coclass "Polyline" :

```
Dim pPointColl As IPointCollection
  Set pPointColl = New Polyline
  pPointColl.AddPoint pPoint1
  pPointColl.AddPoint pPoint2
  pPointColl.AddPoint pPoint3
  pPointColl.AddPoint pPoint4
```



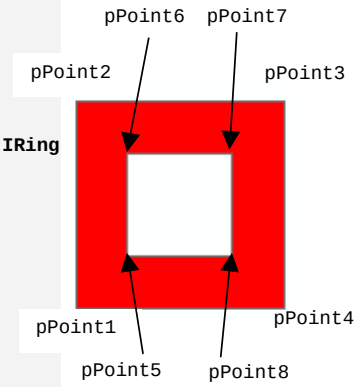
6.2.4. Rings et Polygons

Un ring est la collection d'un ou plusieurs paths qui ferment. Un polygone est composé d'un ou plusieurs rings.



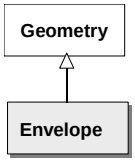
L'exemple suivant décrit la construction d'un polygone composé de deux "rings", elles mêmes construites à partir de "lines" (on suppose que les points utilisés ont été construits au préalable).

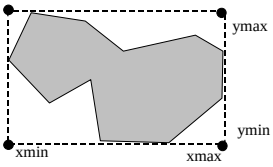
```
Dim pSegColla As ISegmentCollection
Set pSegColla = New Ring
Dim pLine As ILine
Set pLine = New Line
pLine.PutCoords pPoint1, pPoint2
pSegColla.AddSegment pLine
Set pLine = New Line
pLine.PutCoords pPoint2, pPoint3
pSegColla.AddSegment pLine
Set pLine = New Line
pLine.PutCoords pPoint3, pPoint4
pSegColla.AddSegment pLine
Dim pRinga As IRing
Set pRinga = pSegColla 'QI entre ISegmentCollection et IRing
pRinga.Close 'Fermeture du polygone

Dim pSegCollb As ISegmentCollection
Set pSegCollb = New Ring
Set pLine = New Line
pLine.PutCoords pPoint5, pPoint6
pSegCollb.AddSegment pLine
Set pLine = New Line
pLine.PutCoords pPoint6, pPoint7
pSegCollb.AddSegment pLine
Set pLine = New Line
pLine.PutCoords pPoint7, pPoint8
pSegCollb.AddSegment pLine
Dim pRingb As IRing
Set pRingb = pSegCollb 'QI
pRingb.Close

Dim pPolygon As IGeometryCollection
Set pPolygon = New Polygon
pPolygon.AddGeometry pRinga
pPolygon.AddGeometry pRingb
```



6.2.5. Envelopes

L'enveloppe d'une géométrie (au sens large) est le rectangle englobant minimum de cette géométrie. Ce rectangle est défini par ses coordonnées x, y minimum et maximum. La propriété "Envelope" (lecture) est définie dans l'interface IGeometry et permet de récupérer l'enveloppe de n'importe quelle géométrie.

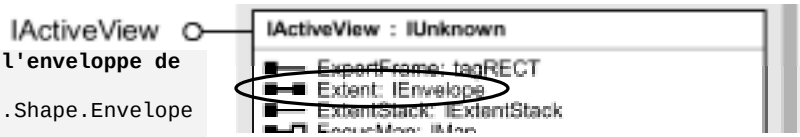




Remarque : une enveloppe étant elle-même un type de géométrie particulier, elle possède également une propriété "envelope" !

Le code ci-dessous montre un exemple d'utilisation de l'enveloppe d'une entité pour zoomer sur cette entité :

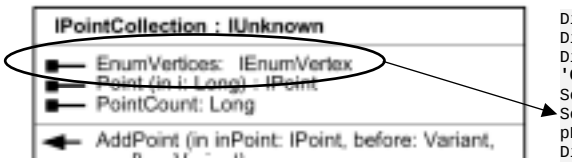
```
'L'étendue de la carte est égale à l'enveloppe de
'l'entité pFeature
pMxDoc.ActiveView.Extent = pFeature.Shape.Envelope
```

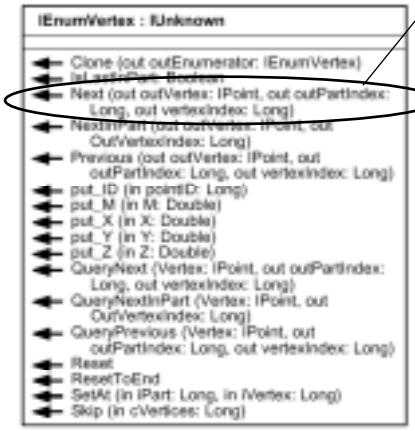


6.3. EXEMPLE : LECTURE DE LA GEOMETRIE D'UN POLYGONE

La propriété "EnumVertices" de l'interface IPointCollection retourne un énumérateur des points de la collection. Cette interface "IEnumVertex" possède un certain nombre de méthodes pour lire (et écrire) les coordonnées d'une géométrie même complexe. Elle permet entre autre d'avoir des informations sur la partie de géométrie étudiée dans le cas de géométrie multi-parties.

L'exemple ci-dessous affiche dans un MsgBox les coordonnées du polygone construit précédement (6.2.4), ainsi que des informations sur le n° de partie à laquelle appartient le vertex.






```
Dim sEnum As String
Dim pEnum As IEnumVertex
Dim pPointcollection As IPointCollection
'QI entre IGeometryCollection et IPointcollection
Set pPointcollection = pPolygon
Set pEnum = pPointcollection.EnumVertices
pEnum.Reset 'Pour être sûr d'être positionné au début de l'énumérateur
Dim pPoint As IPoint, lPart As Long, lVertex As Long

pEnum.Next pPoint, lPart, lVertex
'Parcours du polygon point par point
Do While Not pPoint Is Nothing
  'On concatène le n° de partie, de vertex et les coordonnées du point
  sEnum = sEnum & "partie n° " & lPart & " - vertex n° " & lVertex _
     & " : X = " & pPoint.X & ", Y = " & pPoint.Y & Chr(13)
 pEnum.Next pPoint, lPart, lVertex
Loop
MsgBox sEnum
```

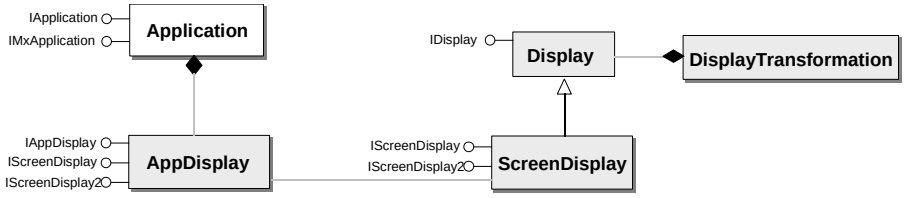
7. Affichage et rafraichissement d'écran

7.1. UTILISATION DU SCREENDISPLAY POUR DESSINER DES ELEMENTS GRAPHIQUES

ArcMap est une application pouvant être composée de plusieurs fenêtres (par exemple la fenêtre principale et la fenêtre "Loupe"). L'application ArcMap est associée à un "**AppDisplay**" qui fournit les outils pour travailler avec plusieurs fenêtres en même temps. Par exemple, l'outil "Déplacer" (pan) utilise l'AppDisplay pour déplacer la vue active ainsi que les vues "Loupe" éventuelles. L'interface "**IScreenDisplay**" implémentée entre autre par AppDisplay fournit des méthodes pour dessiner des éléments graphiques : StartDrawing, FinishDrawing, DrawPoint, DrawPolygon ...etc

Chaque vue se compose en effet d'un objet "**ScreenDisplay**" qui permet d'exécuter des dessins. Le ScreenDisplay est l'écran d'affichage associé à une fenêtre. Il gère les attributs d'affichage propres à l'écran mais également les autres paramètres spécifiques à la fenêtre (cache, défilement...).
L'interface "**IscreenDisplay**" est implémentée par la classe "AppDisplay" ainsi que par la classe "**ScreenDisplay**". On peut également obtenir une référence à un ScreenDisplay via un objet "ActiveView" (Map ou PageLayout) ou bien un objet "MapInsetWindow" (fenêtre loupe), afin de contrôler la fenêtre associée à chacun d'eux.

Enfin, chaque ScreenDisplay est associé à un objet "DisplayTransformation qui gère les passages d'un ScreenDisplay à un autre (cf chap. 11).

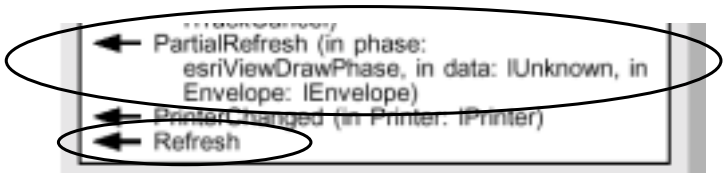


Le code VBA suivant montre comment dessiner rapidement une géométrie donnée (pPolygon qu'on suppose créé auparavant) et suivant un symbole donné (pSym qu'on suppose créé auparavant), en utilisant IScreenDisplay. Ce type d'éléments graphiques n'est pas enregistré, il disparait lorsqu'on rafraichit l'écran. On utilise dans un premier temps la propriété "Display" sur IMxApplication, qui renvoie un IAppDisplay. Après une Query Interface, on récupère un IScreenDisplay afin d'utiliser les méthodes de dessin :

```
Dim pMxApp As IMxApplication
Set pMxApp = Application 'QI entre IApplication et IMxApplication
Dim pAppDisplay as IAppDisplay
set pAppDisplay = pMxApp.Display
Dim pSDisplay As IScreenDisplay
Set pSDisplay = pAppDisplay 'QI entre IAppDisplay et IScreenDisplay
'Début du dessin, dans le device context pSDisplay.hdc (sans cache)
pSDisplay.StartDrawing pSDisplay.hDC, esriNoScreenCache
pSDisplay.SetSymbol pSym
pSDisplay.DrawPolygon pPolygon
pSDisplay.FinishDrawing
```

7.2. RAFRAICHISSEMENTS D'ECRAN

ScreenDisplay, outre les opérations de dessin, permet la création de caches. Un cache est un bitmap représentant la fenêtre de l'application. Au lieu dêtre dessinés directement à l'écran, les éléments graphiques sont dessinés dans des caches qui sont eux-même ensuite dessinés à l'écran. Ainsi, lorsque la fenêtre a besoin d'être rafraichie, le dessin est réalisé à partir des caches au lieu de la base de donnée, ce qui améliore les performances.
En général, l'objet "Map" crée au moins trois caches : un pour l'ensemble des couches, un autre pour les annotations et éléments graphiques et enfin un troisième pour les entités sélectionnées.



IActiveView possède deux méthodes "PartialRefresh" et "Refresh" permettant de rafraichir l'écran. "Refresh" Invalide tous les caches, ce qui le rend peu performant, alors que "PartialRefresh" utilise au maximum les caches (il est donc à utiliser de préférence).
La méthode "PartialRefresh" prend comme paramètres en entrée la "phase" qui détermine le ou les cache(s), et les données ou un rectangle spécifique à invalider (c'est à dire à redessiner).
Les méthodes "PartialRefresh" et "Refresh" font toutes les deux appel à la méthode "Invalidate" de IScreenDisplay, dont les paramètres sont similaires (l'utilisation en est plus délicate car il faut lui préciser quel cache mettre à jour, suivant la phase choisie).

Les différentes phases possibles sont les suivantes :

```
0 - esriViewNone
1 - esriViewBackground   - esriAllScreenCaches
2 - esriViewGeography
4 - esriViewGeoSelection
8 - esriViewGraphics
16 - esriViewGraphicSelection
32 - esriViewForeground
```

Les exemples suivants montrent comment utiliser les phases de "PartialRefresh" suivant qu'on est en mode données ou bien en mode mise en page :

Mode données (Map) :
- Re-dessin d'une couche pLayer donnée :

```
pActiveView.PartialRefresh esriViewGeography, pLayer, Nothing
```

- Re-dessin de toutes les couches :

```
pActiveView.PartialRefresh esriViewGeography, Nothing, Nothing
```

- Re-dessin de la sélection :

```
pActiveView.PartialRefresh esriViewGeoSelection, Nothing, Nothing
```

- Re-dessin des étiquettes :

```
pActiveView.PartialRefresh esriViewGraphics, Nothing, Nothing
```

Mode mise en page (PageLayout) :

- Re-dessin d'un élément pElement donné :

```
pActiveView.PartialRefresh esriViewGraphics, pElement, Nothing
```

- Re-dessin de tous les éléments :

```
pActiveView.PartialRefresh esriViewGraphics, Nothing, Nothing
```

- Re-dessin de la sélection :

```
pActiveView.PartialRefresh esriViewGraphicSelection, Nothing, Nothing
```

Plusieurs phases peuvent être combinées, par exemple pour invalider à la fois les couches (esriViewGeography = 2) et la sélection (esriViewGeoSelection = 4) :

```
pActiveView.PartialRefresh   esriViewGeography   +   esriViewGeoSelection,
Nothing, Nothing
```

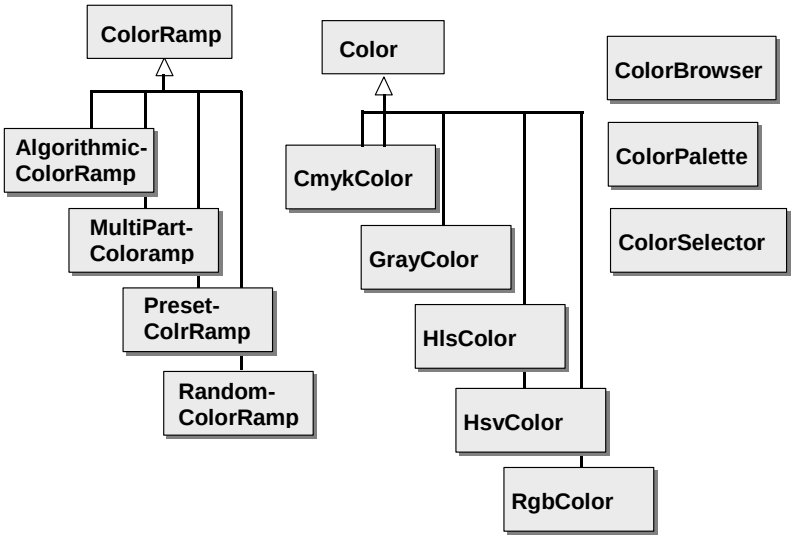
ce qui équivaut à :

```
pActiveView.PartialRefresh 6, Nothing, Nothing
```

8. Symboles

8.1. COULEUR

Les classes nécessaires à la manipulation de couleurs avec ArcObjects sont les suivantes :

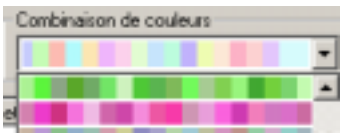


A chacune des 5 sous-classes de couleur correspond une interface :
ICmykColor : Cyan, magenta, yellow, black (0-255)
IGrayColor : Level 0 (white) à 255 (black)
IHlsColor : Hue, lightness, and saturation (0-100)
IHsvColor : Hue (0-360), saturation and value (0-100)
IRgbColor : Red, green, and blue (0-255)

Le code VBA ci-dessous montre comment créer une couleur de type RGBColor et l'affecter à un symbol (pSymbol à créer au préalable) :

```
dim pRgbColor as IRgbColor
set pRgbColor as New RgbColor
pRgbColor.Red = 200
pRgbColor.Green =100
pRgbColor.Blue = 57
set pSymbol.color = pRgbColor
```

Une **ColorRamp** est une collection de couleurs :



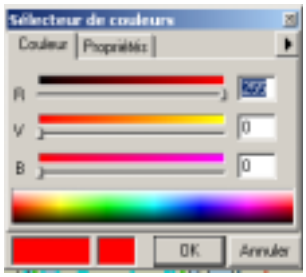
Il existe quatre types de ColorRamp :
- AlgorithmicColorRamp : dégradé de couleur entre une "from" color et une "to" color
- MultiPartColorRamp : Ramp composée de plusieurs ColorRamp
- PresetColorRamp : collection de 13 couleurs prédéfinies
- RandomColorRamp : série de couleurs aléatoires entre une "from" color et une "to" color

Voir dans le § 9.2.3 un exemple de création d'une AlgorithmicColorRamp.

ColorBrowser, ColorPalette et ColorSelector correspondent aux différentes interfaces utilisateur que l'on peut rencontrer pour choisir une couleur sous ArcMap.

Le code VBA suivant montre l'utilisation de la classe ColorSelector pour faire apparaître à l'écran le sélecteur de couleurs ArcMap. La couleur sélectionnée par l'utilisateur (propriété color) est ensuite utilisée dans cet exemple comme couleur de sélection de la couche pFLayer (propriété SelectionColor de IFeatureSelection) :

```
Dim pColor As IColor
Set pColor = New RgbColor
pColor.RGB = 255 'Rouge
Dim pFSel As IFeatureSelection
Set pFSel = pFLayer 'QI
pFSel.SetSelectionSymbol = False

Dim pselector As IColorSelector
Set pselector = New ColorSelector
pselector.Color = pColor
Dim bColorSel As Boolean
bColorSel = pselector.DoModal(0)
If bColorSel Then Set pFSel.SelectionColor = pselector.Color
```



Remarque : noter la différence avec l'exemple précédent utilisant l'interface IRgbColor où la couleur est définie par trois propriété Red, Green, Blue. L'interface IColor possède une propriété "RGB" qui définit une couleur par un entier long. On peut utiliser la fonction VB "RGB" suivante pour définir une couleur :

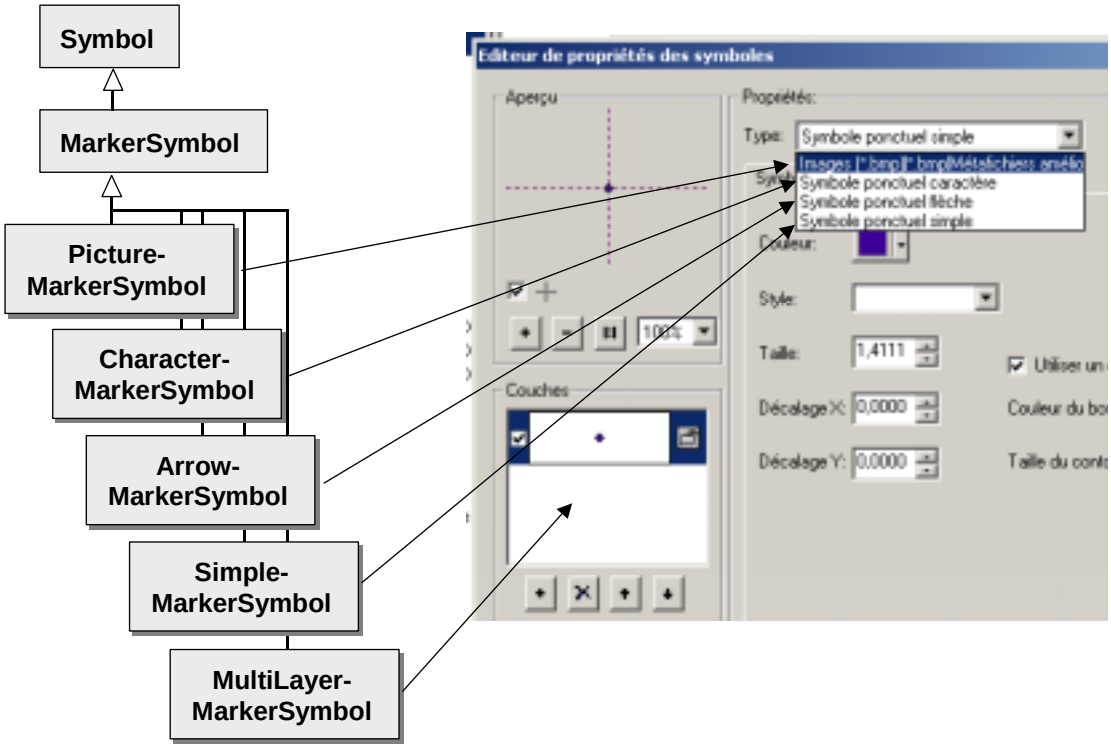
```
ColMyColor.RGB = RGB(intMyRedValue, intMyGreenValue, intMyBlueValue)
```

ou bien la fonction suivante qui permet de voir comment est faite la conversion :

```
Public Function RGBToLong(lngRed as Long, lngGreen as Long, lngBlue as Long) as Long
  RGBToLong = lngRed + (&H100 * lngGreen) + (&H10000 * lngBlue)
End Function
```

8.2. POINTS

Les classes ArcObjects permettent de définir par programme l'équivalent de ce qu'on peut faire avec l'éditeur de propriétés des symboles d'ArcMap :

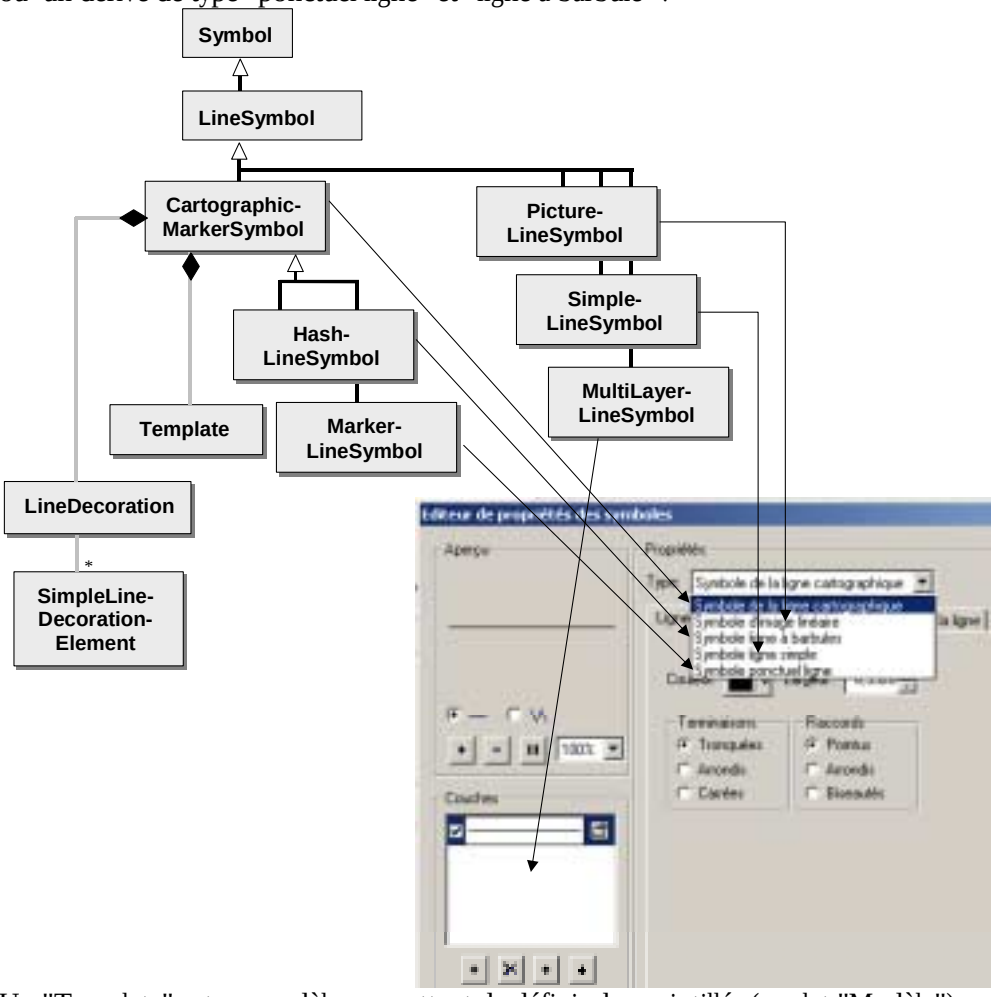


Le code VBA suivant montre comment comment créer par exemple un "PictureMarkerSymbol" à partir d'une image Bitmap :

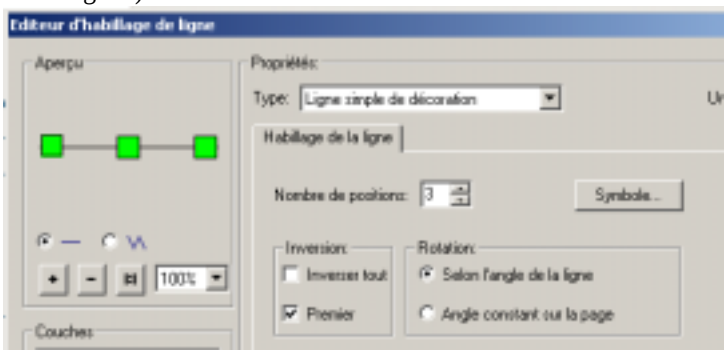
```
Dim pPictMarker As IPictureMarkerSymbol
Set pPictMarker = New PictureMarkerSymbol
pPictMarker.CreateMarkerSymbolFromFile esriIPictureBitmap, _
                                       "d:\ENSG_coul26mm-300dpi.bmp"
```

8.3. LIGNES

Le modèle pour les symboles linéaires est plus complexe et reflète le nombre important de possibilité de l'éditeur de symboles linéaires, notamment lorsqu'on crée un "symbole de ligne cartographique" ou un dérivé de type "ponctuel ligne" et "ligne à barbule" :

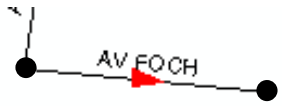


Un "Template" est un modèle permettant de définir des pointillés (onglet "Modèle").
LineDecoration et SimpleLineDecoration permettent de définir des "Ligne simple de décoration" (onglet "Propriétés de la ligne") :



Les "LineDecoration" appartiennent à "CartographicLineSymbol" et ne sont pas des symboles en tant que tels. Ils sont utilisés pour placer des "décorations" tels que des flèches à des emplacements particuliers le long de la ligne (par exemple au début ou à la fin).

ArcMap ne permet pas en standard de créer un seul symbole "flèche" au milieu d'une ligne (et non pas au début et à la fin). Le code VBA ci-dessous crée ce type de symbole :

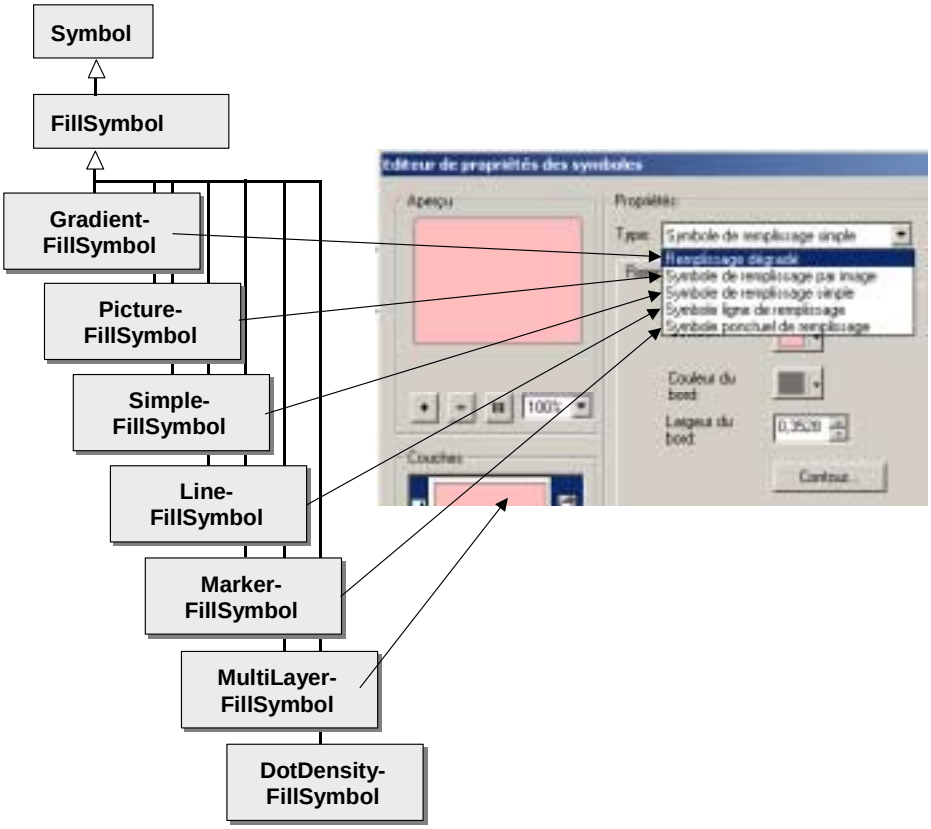


On utilise pour cela l'interface "ILineProperties" implémentée par la classe "CartographicLineSymbol" qui possède une propriété "LineDecoration" qui permet d'associer une "LineDecoration" au "CartographicLineSymbol" (c'est également cette interface qu'il faut utiliser pour implémenter des symboles à barbules, tiretés ou ponctuels ligne).
Le principe consiste à créer un symbole ponctuel (de type flèche), puis créer un "SimpleLineDecorationElement" à partir de ce symbole, en paramétrant le positionnement de celui-ci à la moitié de la ligne (grâce à la propriétés "PositionAsRatio" et à la méthode "AddPosition"). On ajoute ensuite cet élément de décoration à un nouvel objet LineDecoration, que l'on associe ensuite au symbole de ligne cartographique.

```
'Création d'un nouveau symbole de ligne cartographique
Dim pLineProp As ILineProperties
Set pLineProp = New CartographicLineSymbol

'Création d'une nouvelle LineDecoration
Set pLineProp.LineDecoration = New LineDecoration

'Création d'une couleur pour le symbole ponctuel
Dim pColor As IColor
Set pColor = New RgbColor
pColor.RGB = 255 'rouge

'Création d'un symbole ponctuel de type flèche
Dim pMarker As IArrowMarkerSymbol
Set pMarker = New ArrowMarkerSymbol
pMarker.Style = esriAMSPlain
pMarker.Size = 10
pMarker.Color = pColor

'Création d'un nouveau SimpleLineDecorationElement
Dim pSLDecorElem As ISimpleLineDecorationElement
Set pSLDecorElem = New SimpleLineDecorationElement
'On lui affecte le symbole flèche et on fixe sa position
With pSLDecorElem
   .MarkerSymbol = pMarker
   .PositionAsRatio = True
   .AddPosition 0.5 'un seul symbole, à la moitié de la ligne
End With

'On ajoute le SimpleLineDecorationElement à la LineDecoration
'et on associe celle-ci au symbole de ligne cartographique
 pLineProp.LineDecoration.AddElement pSLDecorElem
```

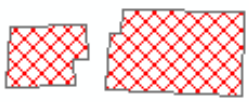
8.4. SURFACE

Les symboles surfaciques mettent en oeuvre les classes suivantes (plus éventuellement celles des symboles ponctuels et linéaires pour les contours et les poncifs) :



Tout comme pour ls symboles ponctuels ou linéaires, les symboles surfaciques possèdent une propriété "Style" avec des valeurs prédéfinies (énumération ESRI). Pour un symbole de remplissage simple, les valeurs sont les suivantes : esriSFSSolid (c'est le style disponible dans l'éditeur de symbole ArcMap), esriSFSHollow, esriSFSHorizontal, esriSFSVertical, esriSFSForwardDiagonal, esriSFSBackwardDiagonal, esriSFSCross et esriSFSDiagonalCross.
L'exemple VBA suivant montre l'utilisation de ce type de symbole avec le style esriSFSDiagonalCross :

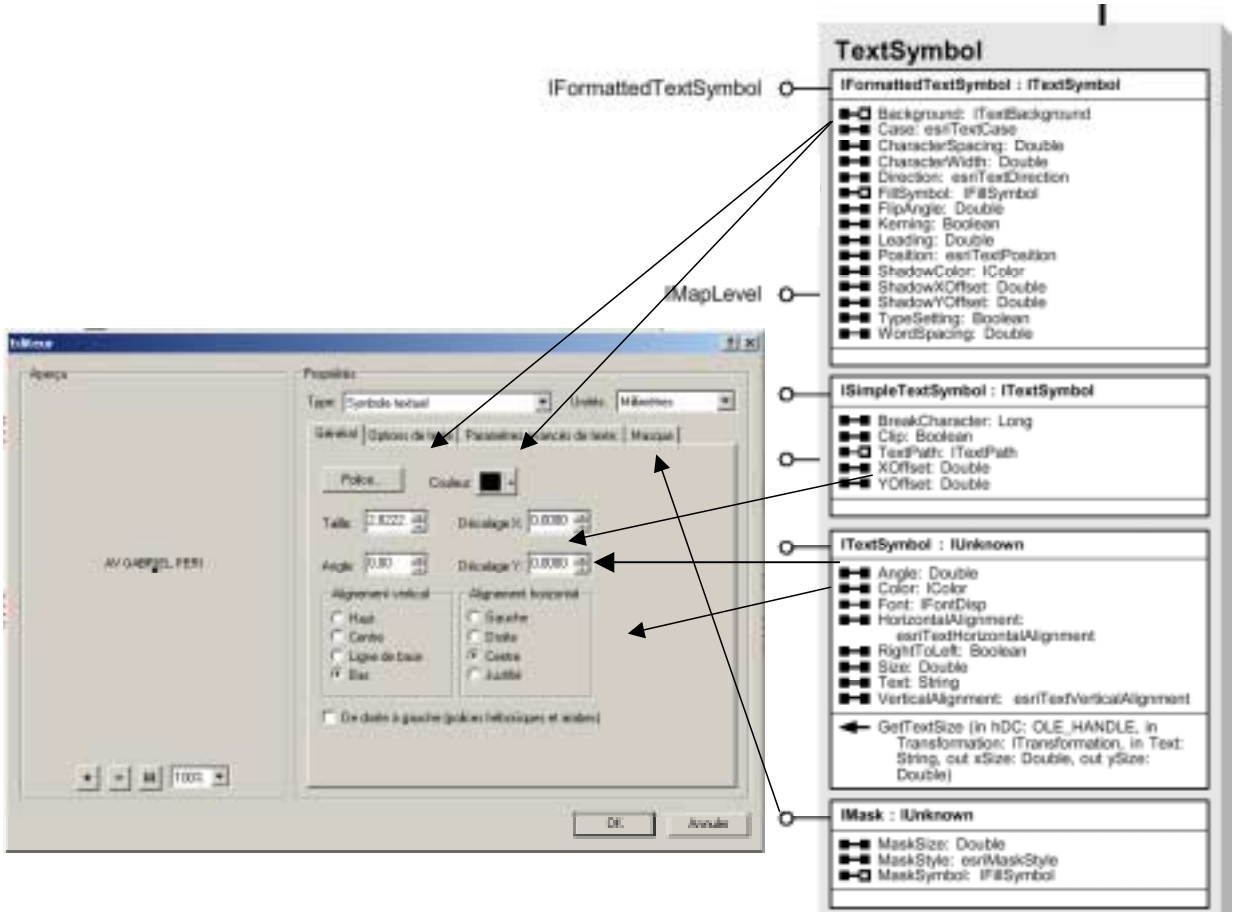
```
Dim pFillSymbol As ISimpleFillSymbol
Set pFillSymbol = New SimpleFillSymbol
pFillSymbol.Style = esriSFSDiagonalCross
pFillSymbol.Color = pColor 'pColor créé avant
'Outline par défaut (ie trait noir de 1pt)
```



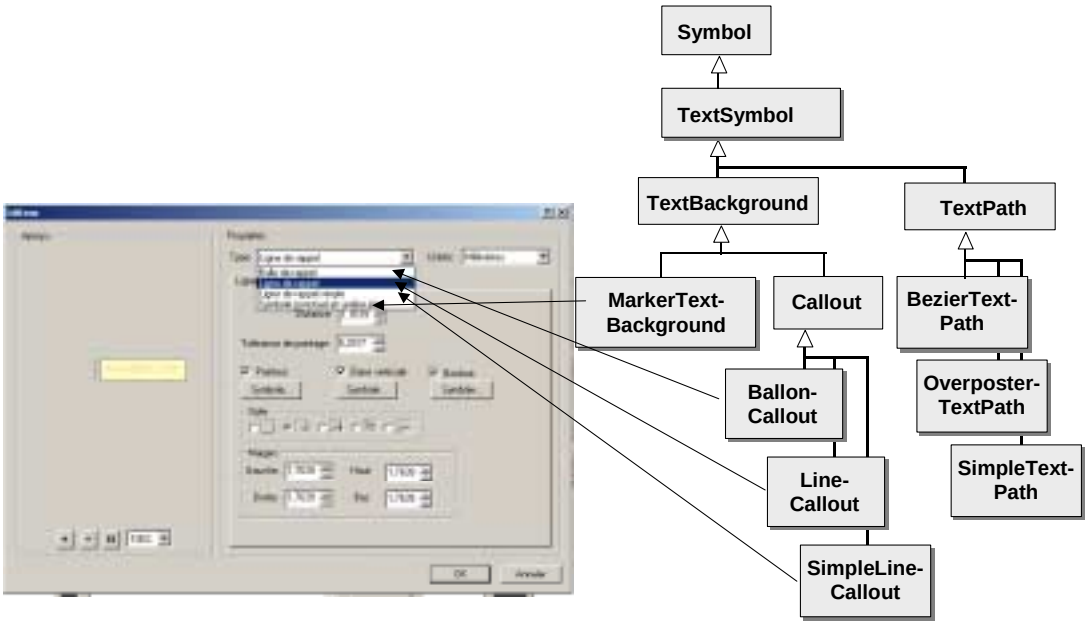
8.5. TEXTE

Il n'existe qu'un seul type de symbole textuel, la classe "TextSymbol" n'est donc pas une Abstract Class. Elle implémente 3 interfaces, ITextSymbol, ISimpleTextSymbol et IFormattedTextSymbol qui permettent de contrôler exactement la manière dont le texte doit apparaître.

On retrouve dans ces trois interfaces les possibilités de l'éditeur de texte ArcMap (+ l'interface IMask permettant de définir un masque autour du texte) :



"TextBackGround" est la classe qui permet de définir des bulles de rappel et des arrière-plans de symboles, on retrouve ces paramètres dans les propriétés "d'Arrière-plan du texte" de l'onglet "Propriétés avancées du texte". TextPath est utilisée pour placer du texte le long d'une courbe :



Pour définir un symbole textuel, la première chose à faire est de définir uen police ("Font"). Il faut pour cela créer un objet COM "Font" grâce à l'interface "IFontDisplay" (bibliothèque "stdole" : Standard OLE COM Library), par exemple un objet de la classe "StdFont" :

```
'Création d'une Font "Arial Black", caractères gras, taille 40
Dim pFont As stdole.IFontDisp
Set pFont = New stdole.StdFont
pFont.Name = "Arial Black"
pFont.Size = 40
pFont.Bold = True
```

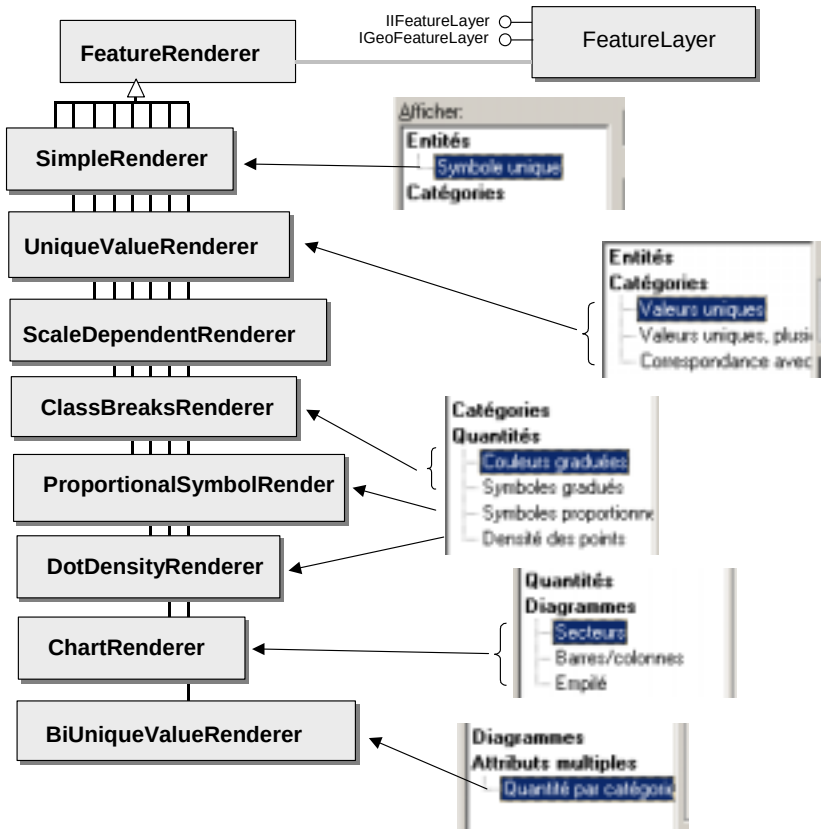
Le code VBA ci-dessous montre l'utilisation de l'interface IFormattedTextSymbol pour créer un symbole texte avec un remplissage du texte par motif dégradé :

```
Dim pTextSymbol As IFormattedTextSymbol
Set pTextSymbol = New TextSymbol
pTextSymbol.Font = pFont
'Création du symbole de remplissage"dégradé"
Dim pGradFillSymb As IGradientFillSymbol
Set pGradFillSymb = New GradientFillSymbol
pGradFillSymb.Style = esriGFSLinear
'On affecte ce symbole au texte
Set pTextSymbol.FillSymbol = pGradFillSymb
```

9. Symbolisation de couches

9.1. LA CLASSE FEATURERENDERER

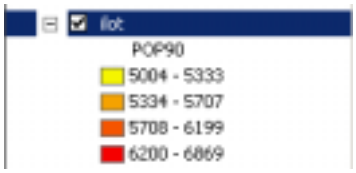
La classe "FeatureRenderer" est la structure qui permet de reproduire par programme le type de symbolisation paramétrable avec l'onglet "Symbologie" sur les propriétés d'une couche. Les sous-classes de FeatureRenderer correspondent aux divers modes de représentation :



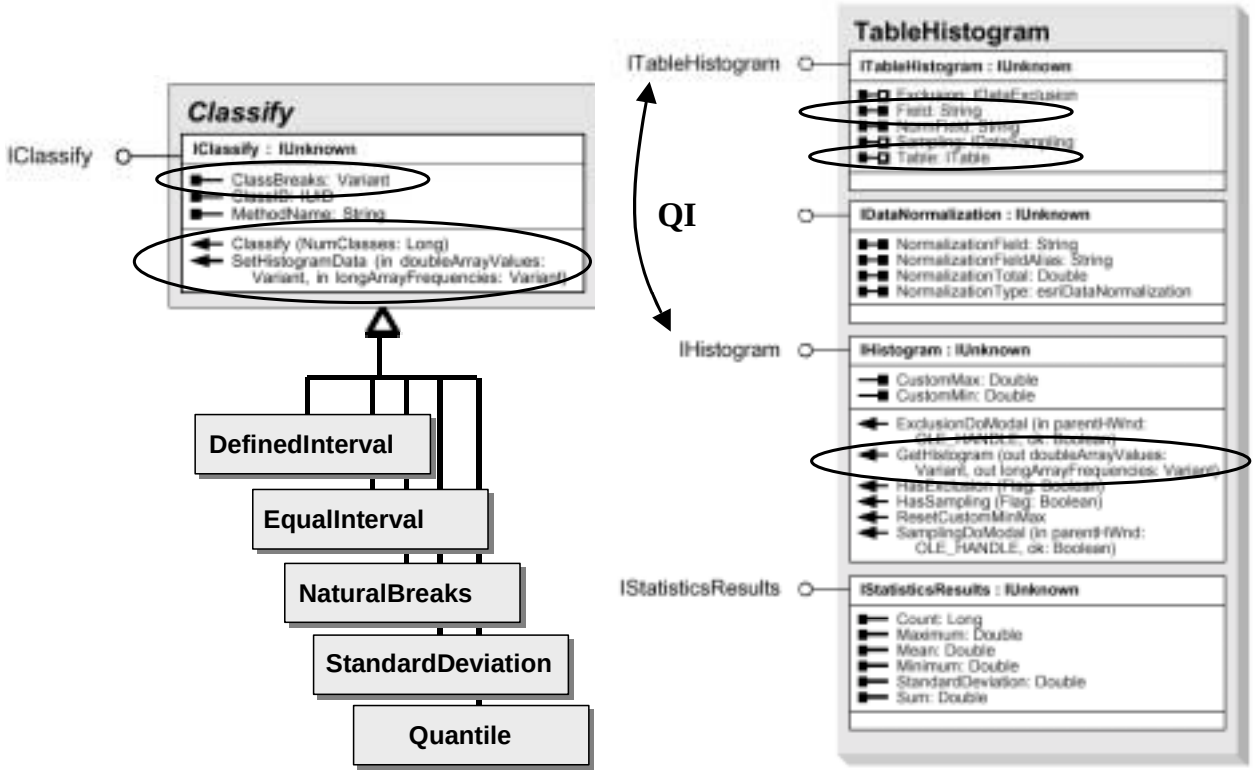
A noter enfin la possibilité de créer son propre type de renderer, voir les exemples de l'aide en ligne (ArcObjetcs Developer Help / Samples / ArcMap / Symbology / Renderers).

9.2. EXEMPLE : REPRESENTATION EN UTILISANT UNE CLASSIFICATION PAR INTERVALLES EGAUX

Il s'agit, par programme, de symboliser une couche de polygones suivant la valeur d'un champ numérique. Le programme ci-dessous représente le champ "POP90" de la couche "ilot" par qautre plages de couleurs suivant une classification par intervalles égaux :



9.2.1. Création de l'histogramme



Pour créer une classification, il faut dans un premier temps récupérer les valeurs de l'histogramme étudié (valeurs et fréquences). Pour cela, on utilise un "TableHistogram", qui est la structure de données permettant d'accéder à ces informations grâce à la méthode "GetHistogram" :

```
Dim pMxDoc As IMxDocument, pFLayer As IFeatureLayer
Set pMxDoc = ThisDocument
Set pFLayer = pMxDoc.FocusMap.Layer(4) '
'QI entre ITable et IFeatureLayer pour accéder à la table de la couche étudiée
Dim pTable As ITable
Set pTable = pFLayer
'Création d'un nouveau TableHistogram
Dim pTH As ITableHistogram
Set pTH = New TableHistogram
'On fait pointer pTH sur la table étudiée et sur le champ "POP90"
Set pTH.Table = pTable
pTH.Field = "POP90"
'QI entre ITableHistogram et IHistogram
Dim pHist As IHistogram
Set pHist = pTH
'On lit les valeurs et fréquences pour le champs "POP90"
Dim dataValues As Variant, dataFrequency As Variant
pHist.GetHistogram dataValues, dataFrequency
```

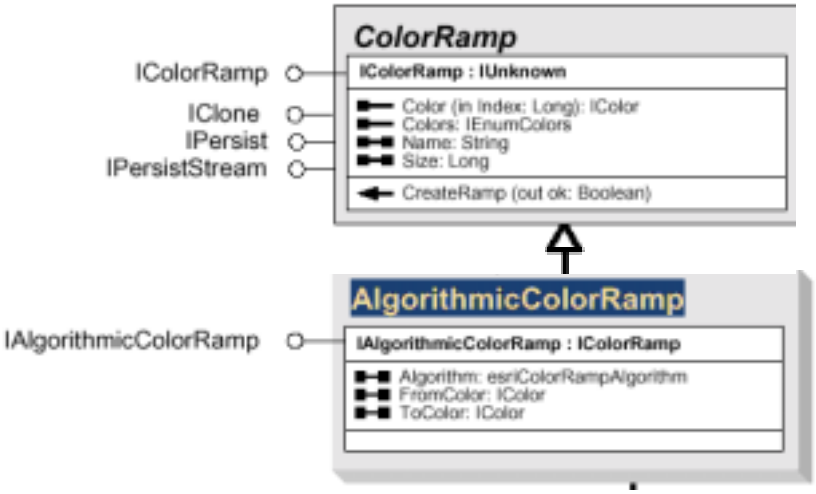
9.2.2. Création de la classification

A partir de l'histogramme, on va pouvoir calculer la classification, c'est à dire calculer les bornes de chaque intervalle et stocker ces valeurs dans un tableau de réels. On crée pour cela un nouvel objet "IClassify" de la sous-classe correspondant à la méthode de classification souhaitée. La méthode "SetHistogramData" récupère la valeur de l'histogramme. La méthode "Classify" exécute la classification puis la méthode "ClassBreaks" permet de remplir le tableau des bornes de classes.

```
' On écrit ces valeurs dans un objet Classify de type EqualInterval
Dim pClassify As IClassify
Set pClassify = New EqualInterval
pClassify.SetHistogramData dataValues, dataFrequency
'Calcul de la classification, les bornes sont stockées dans un tableau
Dim ClassCount As Long
ClassCount = 4 'Nombre de classes
pClassify.Classify ClassCount
Dim CBArray() As Double
CBArray = pClassify.ClassBreaks
```

9.2.3. Création d'une rampe de couleurs

Pour représenter des classes de valeurs suivant un dégradé de couleurs, on crée un objet "AlgorithmicColorRamp" qui va nous permettre ensuite de parcourir les couleurs via un énumérateur de couleurs ("IEnumColors") :

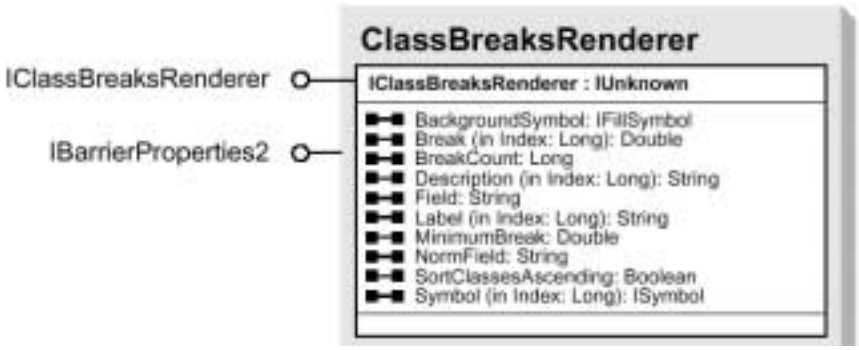


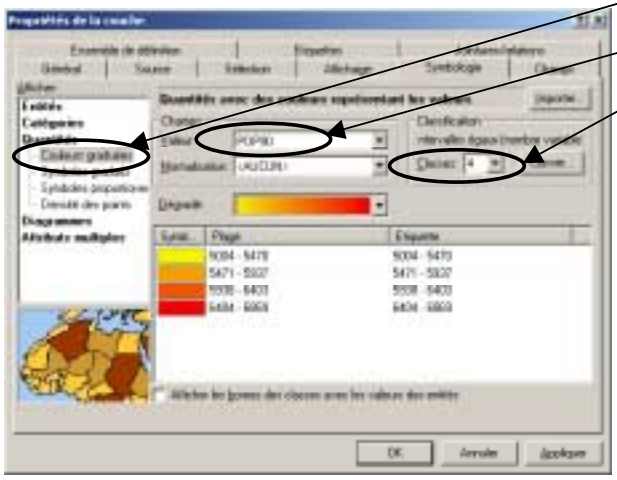
Pour créer la ColorRamp, il est nécessaire de spécifier la couleur de départ, la couleur d'arrivée, l'algorithme utilisé ainsi que le nombre de classes :

```
'On crée un dégradé de couleurs (jaune à rouge)
Dim pFromColor As IRgbColor    'Couleur de départ = jaune
Set pFromColor = New RgbColor
pFromColor.Red = 255
pFromColor.Green = 255
pFromColor.Blue = 0
Dim pToColor As IRgbColor
Set pToColor = New RgbColor   'Couleur d'arrivée = rouge
pToColor.Red = 255
pToColor.Green = 0
pToColor.Blue = 0
'Création de la rampe de couleur
Dim pRamp As IAlgorithmicColorRamp
Dim ok As Boolean
Set pRamp = New AlgorithmicColorRamp
pRamp.Algorithm = esriHSVAlgorithm 'algorithme TSL
pRamp.FromColor = pFromColor
pRamp.ToColor = pToColor
pRamp.Size = ClassCount 'ClassCount=4 couleurs
pRamp.CreateRamp ok
'Création d'une énumération de couleurs
Dim pEnumColors As IEnumColors
Set pEnumColors = pRamp.Colors
```

9.2.4. Création du Renderer

A ce stade, il ne reste plus qu'à créer le Renderer en lui spécifiant ses propriétés, que l'on retrouve dans la fenêtre de paramétrage d'ArcMap. Pour affecter un symbole à chaque classe, on utilise une boucle sur le nombre de classes et on fait appel au tableau créé précédemment pour spécifier les bornes des classes au renderer (propriété "Break") :





NB : l'élément 0 du tableau de la classification correspond au "MinimumBreak" du Renderer ... le "Break(0) du Renderer correspond donc au deuxième élément du tableau, c'est à dire CBArray(1) et aini de suite ..

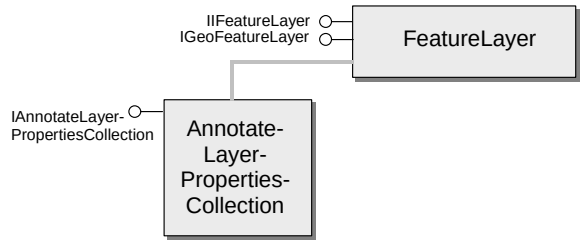
```
'On initialise un nouveau ClassBreakRenderer
Dim pCBRenderer As IClassBreaksRenderer
Set pCBRenderer = New ClassBreaksRenderer
'Paramétrage du Renderer
pCBRenderer.Field = "POP90"
pCBRenderer.MinimumBreak = CBArray(0)
pCBRenderer.BreakCount = ClassCount
Dim pColor As IColor
Dim pFillSymbol As ISimpleFillSymbol
Dim i As Long
For i = 0 To ClassCount - 1
  'pour chaque classe, on affecte
  'un symbole à plat de couleur
  Set pColor = pEnumColors.Next
  Set pFillSymbol = New SimpleFillSymbol
  pFillSymbol.Color = pColor
  pFillSymbol.Style = esriSFSSolid
  pCBRenderer.Symbol(i) = pFillSymbol
  pCBRenderer.Break(i) = CBArray(i + 1)
Next i
'on affecte le renderer à la couche
Dim pGeoFeatureLayer As IGeoFeatureLayer
'QI entre IFeatureLayer et IGeoFeatureLayer
Set pGeoFeatureLayer = pFLayer
Set pGeoFeatureLayer.Renderer = pCBRenderer
'Rafraichissement de la vue
pMxDoc.ActiveView.Refresh
' ... et de la table des matières
pMxDoc.UpdateContents
```

Enfin, on utilise la propriété "Renderer" de IGeoFeatureLayer (nécessité de faire une QueryInterface en IFeatureLayer et IGeoFeatureLayer) pour affecter le Renderer ainsi créé à la couche.

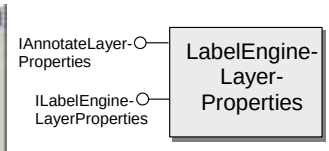
9.3. AFFICHAGE DE LABELS

ArcMap fournit un large panel de possibilités pour afficher des étiquettes sur les entités (filtre sur les entités à étiqueter, contenu de l'étiquette résultat d'une expression VB Script, résolution de conflits d'affichage suivant des pondérations, diverses options de placement pour les points et les lignes ...). ArcObjects permet de reproduire ce type de paramétrage par programme mais également de construire son propre moteur de placement d'étiquette.

L'exemple VBA ci-dessous affiche le contenu du champ "NOM_RUE_G" pour chaque entité (linéaire)de la couche n°2 du document. Pour davantage de précisions sur les méthodes et propriétés, se référer au diagramme "Labeling and Annotation OMD" d'ArcObjects.




```
'Récupération de la couche à étiqueter
Dim pMxDoc As IMxDocument, pFLayer As IFeatureLayer
Set pMxDoc = ThisDocument
Set pFLayer = pMxDoc.FocusMap.Layer(2)
Dim pGeoFLayer As IGeoFeatureLayer
Set pGeoFLayer = pFLayer 'QI
'On récupère l'ensemble des propriétés d'étiquetage
' de la couche pGeoFLayer, ce qui correspond à
'l'onglet étiquettes des propriétés de la couche
Dim pAnnoProps As IAnnotateLayerPropertiesCollection
Set pAnnoProps = pGeoFLayer.AnnotationProperties
```

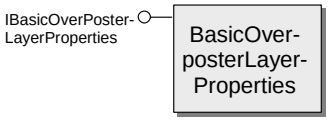



```
'On définit les propriétés d'étiquetage de la
'couche en créant une nouvelle "Classe"
Dim pLabelEngine As ILabelEngineLayerProperties
Set pLabelEngine = New LabelEngineLayerProperties
pLabelEngine.Expression = "[NOM_RUE_G]"
```

```
'On définit un nouvel objet LineLabelPosition
'pour spécifier les options de positionnement
Dim pPosition As ILineLabelPosition
Set pPosition = New LineLabelPosition
pPosition.ProduceCurvedLabels = True
pPosition.Above = True
```

```
'On définit un objet BasicOverposterLayerProperties qui
'supporte les options de placement et conflits, ce qui
'correspond à la fenêtre propriétés de placement
Dim pBasic As IBasicOverposterLayerProperties
Set pBasic = New BasicOverposterLayerProperties
pBasic.FeatureType = esriOverposterPolyline
pBasic.LineLabelPosition = pPosition
pBasic.NumLabelsOption = esriOneLabelPerName
```

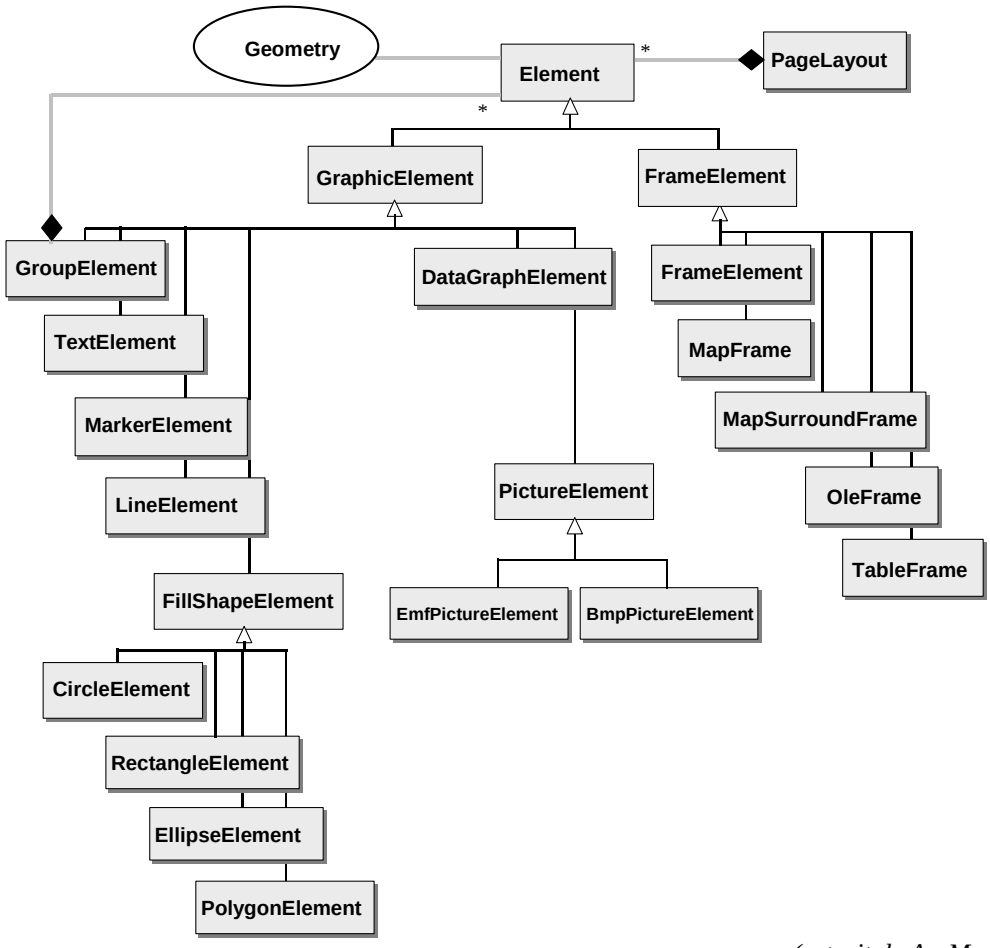
```
'On affecte ces options de placement aux pLabelEngine
Set pLabelEngine.BasicOverposterLayerProperties = pBasic
'QI entre IAnnotateLayerProperties et ILabelEngineLayerProperties
Dim pAnnoLayerProps As IAnnotateLayerProperties
Set pAnnoLayerProps = pLabelEngine
pAnnoProps.Add pAnnoLayerProps 'Ajout de la classe d'étiquette
'Puis affichage des étiquettes et rafraichissement de l'écran
pGeoFLayer.DisplayAnnotation = True
pMxDoc.ActiveView.Refresh
```



Remarques : cet exemple simplifié ne fait pas intervenir de requêtes SQL, ni de paramètres particulier de détection de conflits. Il ne gère pas non plus les classes d'étiquettes existantes ... Après exécution, il existe deux classes nommées "par défaut" ...

10. Mise en page

10.1. SCHEMA GENERAL



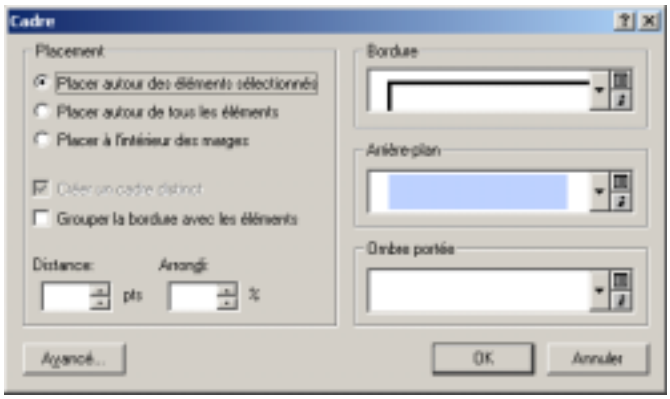
(extrait de ArcMap OMD)

Une mise en page ("PageLayout" cf. § 4.5) est composée de cartes basées sur des entités géographiques et d'éléments autres. La classe "**Element**" décrit tous ces objets, entre autre tous les éléments graphiques tels les textes, points, images, etc ... ainsi que les cadres entourant les cartes et objets liés aux cartes tels les légendes, barres d'échelles etc ... Cette classe possède notament une propriété "Geometry" permettant d'accéder à la géométrie de l'objet (par exemple un élément texte est positionné par un point).

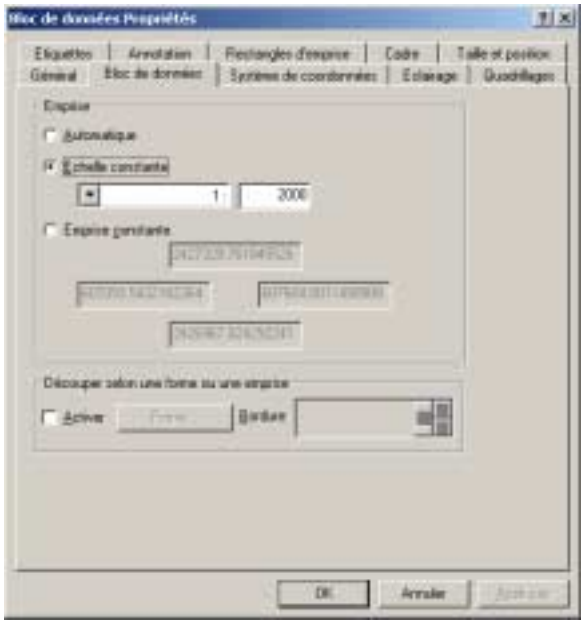
Les "**GraphicElement**" correspondent aux éléments que l'on peut constuire et manipuler avec la barre d'outils "Dessin" d'ArcMap, ainsi que les images et les diagrammes insérés grâce aux menus d'ArcMap :

Un "GroupElement" est un ensemble d'éléments regroupés (de la même manière qu'avec l'item "Grouper" du menu "Dessin".

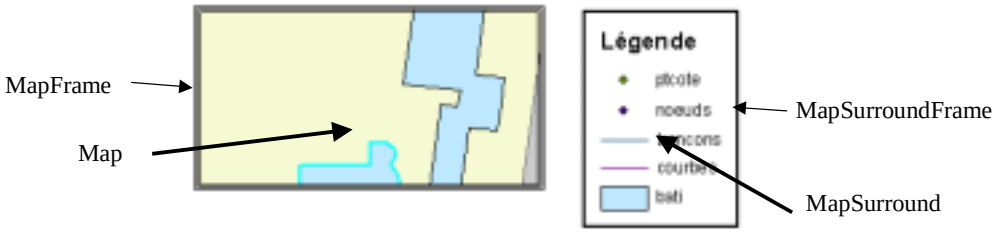
Les "FrameElement" correspondent aux autres éléments de la mise en page, c'est à dire aux bordures et arrière-plans des éléments cartographiquues. La CoClass "**FrameElement**" correspond à l'objet "Cadre" d'ArcMap :



Un "**MapFrame**" est un élément de type FrameElement lié à un bloc de données (Map). Le développeur peut ainsi accéder à l'objet Map via cet élément, ainsi qu'aux propriétés bordure, arrière-plan, quadrillage .... L'interface IMapFrame implémentée uniquement par la CoClass MapFrame permet entre autre de gérer les recatngles d'emprise ("ILocatorRectangle"), l'échelle de la carte (propriétés "MapScale", "MapBounds" et "ExtentType"). On retrouve ces paramétrages dans les propriétés du bloc de données :



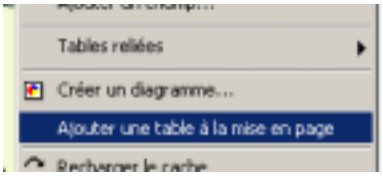
Un "**MapSurroundFrame**" est un élément de type FrameElement lié à un "MapSurround", c'est à dire un éléments cartographique lié au bloc de donnée de type légende, échelle ou flèche du Nord (cf. § suivant). Un MapSurroundFrame décrit le cadre et l'arrière-plan de l'objet "MapSurround". L'interface IMapSurroundFrame fournit deux propriétés MapFrame et MapSurround pour accéder respectivement au MapFrame du bloc de données associé et au MapSurround associé.



NB : la méthode "CreateSurroundFrame" sur IMapFrame permet de créer des éléments liés à la carte tels la flèche du Nord, la légende ....

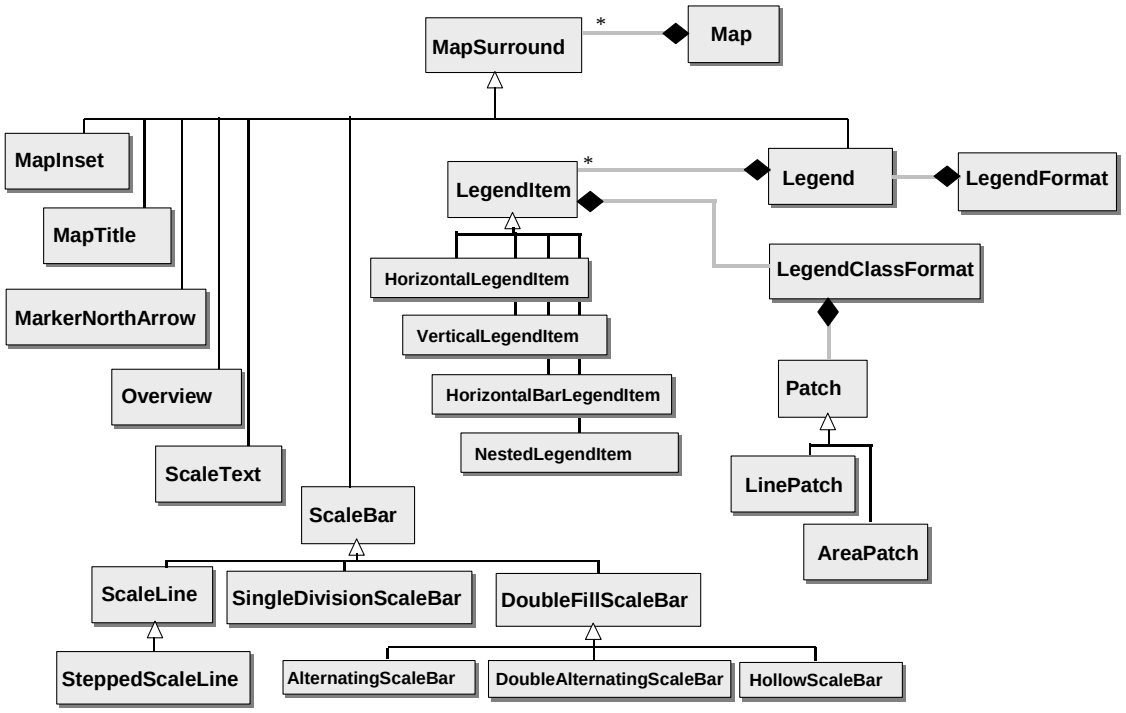
Un "**OleFrame**" fournit les outils nécessaires pour incorporer un objet "OLE", par exemple une table Excel ou un document Word.

Un "**TableFrame**" est une élément correspondant à l'affichage d'une table dans la mise en page via le menu "Options" :



10.2. CAS PARTICULIER DES ELEMENTS LIES A LA CARTE

Les "**MapSurround**" sont des objets spécifiques associés à un objet "Map" (bloc de données). Une flèche du Nord par exemple est un "MapSurround" : si le bloc de données subit une rotation, la flèche du Nord associée subit la même rotation. Les "MapSurround" sont toujours contenus par un "MapSurroundFrame", de même que que les "Map" sont contenus par des "MapFrame" (cf. § précédent). Chaque MapSurroundFrame est également relié à un MapFrame (propriété "MapFrame" de IMapSurroundFrame"). Le "PageLayout" gère les objets "Frame" mais si un bloc de données (Map) est supprimé, ses "MapSurrounds" et "Frames" associés sont supprimés.

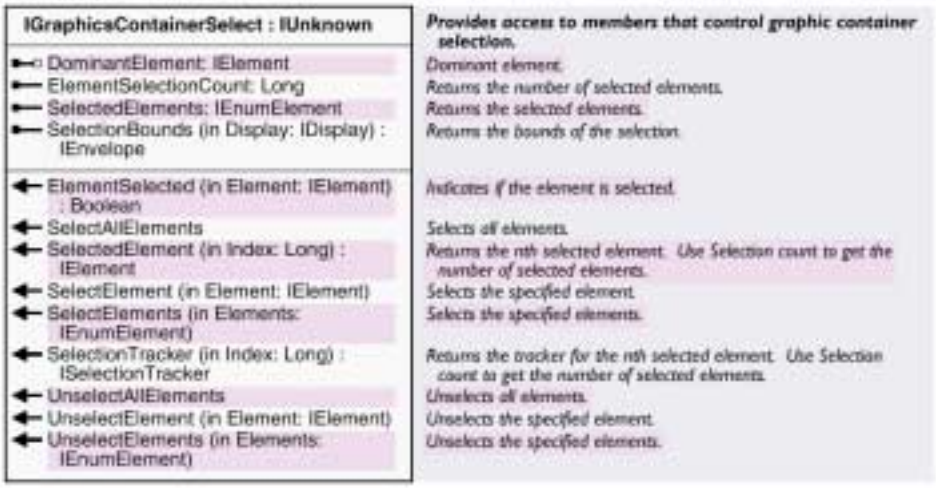


(extrait de ArcMap OMD)

La CoClass "Legend" est particulièrement complexe, du au grand nombre d'options offertes à l'utilisateur lorsqu'il édite une légende. Chaque couche de la carte est représentée par un "LegendItem". Le "LegendItem" gère l'information sur la couche (Layer) associée ainsi que l'aspect de la légende. Chaque "LegendItem" possède un "LegendClassFormat" qui contient des informations optionnelles sur l'aspect de la légende (taille du texte, aspect du patch, ie du gabarit ...). La plupart du temps, ces proriétés ne sont pas définies, l'objet "Legend" utilisant l'objet "LegendFormat" définissant des propriétés par défaut.

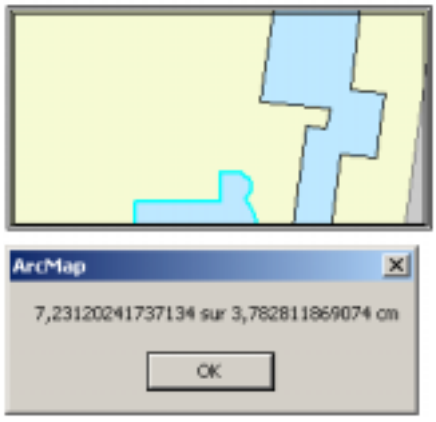
10.3. GRAPHICS CONTAINER

L'interface "IGraphicsContainer" implémentée par la CoClass "PageLayout" permet d'accéder aux divers éléments de la mise en page. Elle fournit notament un certain nombre de méthodes pour ajouter ou supprimer des éléments, gérer l'ordre d'affichage des éléments les uns par rapport aux autres, retirer ou ajouter un élément à un groupe. La méthode "Next" permet de parcourir le GraphicsContainer élément par élément. La position des éléments de la mise en page est définie dans les unités de la page (propriété "Units" de IPage).



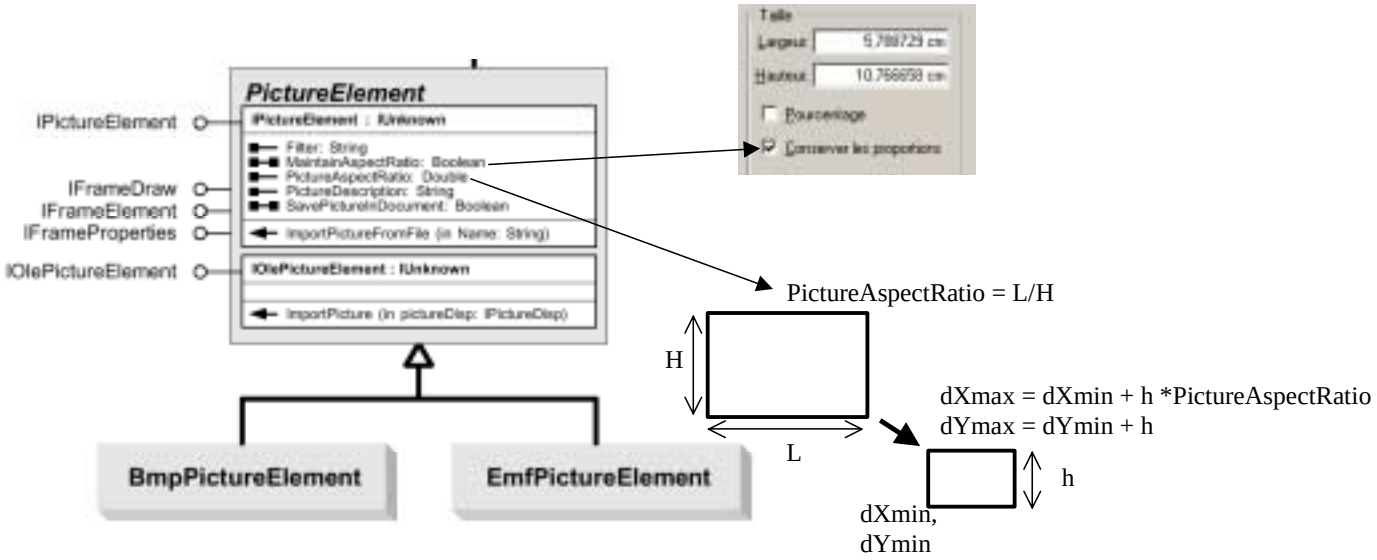
Le code VBA ci-dessous récupère le graphicscontainer associé au document, parcourt l'ensemble des éléments dans une boucle, et lorsque l'élément est de type "MapFrame", affiche dans un MsgBox les dimensions du bloc carte ainsi trouvé (coordonnées "papier") :

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pPageLayout As IPageLayout
'On récupère la mise en page du document
Set pPageLayout = pMxDoc.PageLayout
Dim pGC As IGraphicsContainer
'QI entre IpageLayout et IGraphicsContainer
Set pGC = pPageLayout
pGC.Reset 'Pour être sûr d'être au début
Dim pElem As IElement
Dim pMEnvelope As IEnvelope
'On se positionne sur le premier élément
Set pElem = pGC.Next
Do While (Not pElem Is Nothing)
  'Si l'élément est de type FrameElement
  If (TypeOf pElem Is IFrameElement) Then
    'Si l'élément est de sous-type MapFrame
    If (TypeOf pElem Is IMapFrame) Then
      'On récupère l'enveloppe de l'élément ...
      Set pMEnvelope = pElem.Geometry.Envelope
     '... et on affiche ses dimensions
      MsgBox pMEnvelope.Width & " sur " & pMEnvelope.Height & " cm"
    End If
  End If
  Set pElem = pGC.Next 'On se positionne sur l'élément suivant
Loop
```



10.4. EXEMPLE : AJOUT D'UNE IMAGE A LA MISE EN PAGE

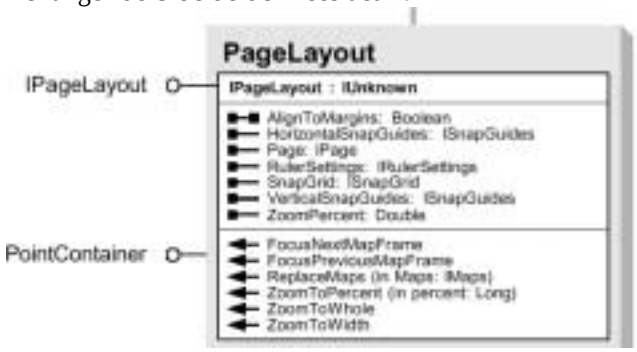
Le code VBA suivant ajoute un élément de type "PictureElement" à la mise en page, en utilisant la propriété "PictureAspectRatio" pour modifier la taille de l'image tout en assurant la conservation des proportions.




```
Dim pMxDoc As IMxDocument, pPageLayout As IPageLayout
Set pMxDoc = ThisDocument
Set pPageLayout = pMxDoc.PageLayout
Dim pGC As IGraphicsContainer
Dim pPic As IPictureElement, pElem As IElement
Set pGC = pPageLayout 'QI entre IPageLayout et IGraphicsContainer
pGC.Reset
Set pPic = New BmpPictureElement 'On crée un nouvel élément image de type bmp
' Import de l'image
pPic.ImportPictureFromFile "D:\ENSG_coul26mm-300dpi.bmp"
pPic.MaintainAspectRatio = True
'Création d'une envelope qui va représenter la géométrie de l'élément image
Dim pEnv As IEnvelope
Dim dXmin As Double, dYmin As Double, dXmax As Double, dYmax As Double
Set pEnv = New Envelope
dXmin = 2
dYmin = 2
dXmax = 2 + (2 * pPic.PictureAspectRatio)
dYmax = 2 + 2
pEnv.PutCoords dXmin, dYmin, dXmax, dYmax
Set pElem = pPic 'QI entre IElement et IPictureElement
pElem.Geometry = pEnv 'la géométrie de l'élément correspond à l'enveloppe
pGC.AddElement pPic, 0 'Ajout de l'élément à la mise en page au niveau 0
Dim pActive As IActiveView
Set pActive = pPageLayout
pActive.Refresh 'Rafraichissement de la mise en page
```

10.5. ZOOM SUR LA PAGE

L'interface IPageLayout fournit des méthodes pour zoomer sur la page (différent du zoom sur les données) ainsi que pour changer de bloc de données actif :

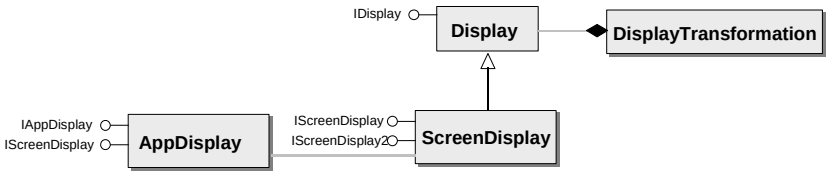


Le code VBA suivant montre par exemple comment zoomer sur la page entière. A noter qu'il est nécessaire de vérifier que l'application est bien en mode "mise en page" (cf. 4.5.1) :

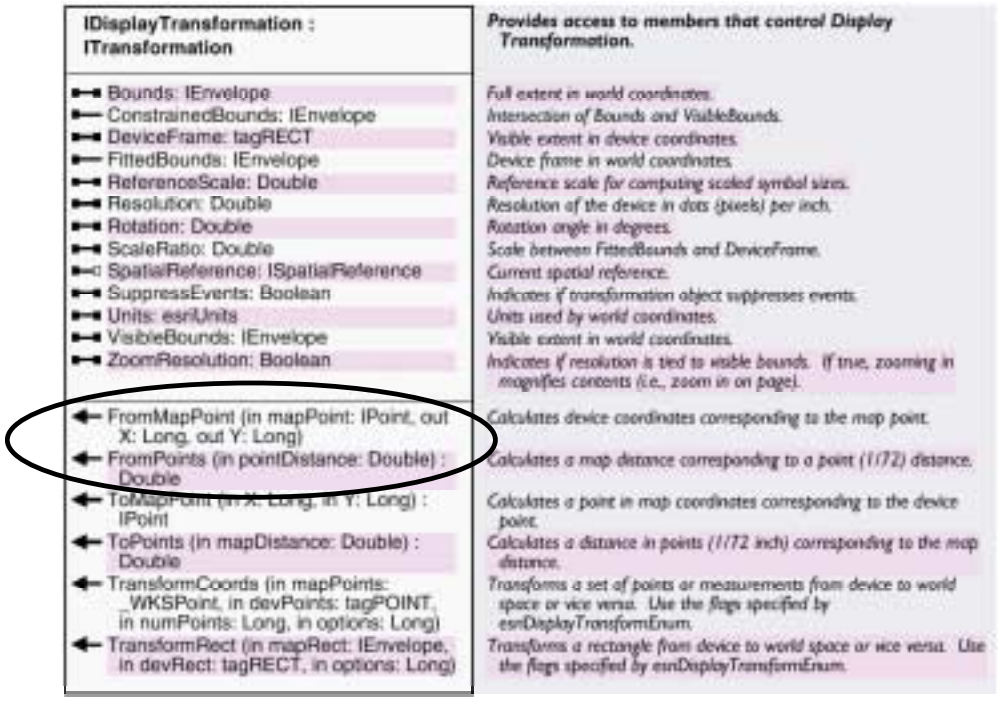
```
Private Sub ZoomToWholePage()
  Dim pMxDoc As IMxDocument, pPageLayout As IPageLayout
  Set pMxDoc = ThisDocument
  Set pPageLayout = pMxDoc.PageLayout
  'Si la vue active n'est pas le mode mise en page
  If Not pMxDoc.ActiveView Is pMxDoc.PageLayout Then
    'Alors on bascule en mode mise en page
    Set pMxDoc.ActiveView = pMxDoc.PageLayout
  End If
  'Zoom sur la page entière
  pPageLayout.ZoomToWhole
End Sub
```

11. Transformation de coordonnées - Projections

11.1. IDISPLAYTRANSFORMATION



(extrait de Display OMD)



Chaque "ActiveView" possède un "ScreenDisplay", lui-même associé à un "DisplayTransformation" (cf. § 7.1). Ce DisplayTransformation permet notamment de passer de coordonnées carte (coordonnées terrain ou page suivant le type d'ActiveView) en coordonnées écran (pixels), ou vice-versa.

Le code VBA suivant transforme un point sur la carte (coordonnées terrain) en point sur la page (coordonnées sur la mise en page) en effectuant successivement deux transformations : passage de coordonnées terrain en coordonnées écran, puis passage des coordonnées écran aux coordonnées page :

```
Private Sub TransformpointFromMapToPageUnits(pPoint As IPoint)
  Dim pMxDoc As IMxDocument
  Set pMxdoc = ThisDocument
  'Récupération des "ActiveView" (PageLayout et Map)
  Dim pLayoutV As IActiveView
  Dim pMapV As IActiveView
  Set pLayoutV = pMxDoc.PageLayout
  Set pMapV = pMxDoc.FocusMap
  'Récupération des DisplayTransformations associés
  Dim pPageTransformation As IDisplayTransformation
  Dim pMapTransformation As IDisplayTransformation
```

```
  Set pPageTransformation = pLayoutV.ScreenDisplay.DisplayTransformation
  Set pMapTransformation = pMapV.ScreenDisplay.DisplayTransformation
  'Passage des coordonnées terrain de pPoint en coordonnées écran
  Dim x As Long
  Dim y As Long
  pMapTransformation.FromMapPoint pPoint, x, y
  'Passage des coordonnées écran en coordonnées page et copie dans pPoint
  Set pPoint = pPageTransformation.ToMapPoint(x, y)
End Sub
```

11.2. CREATION D'UN OUTIL : EXEMPLE D'OUTIL "PAN"

Le "Display" possède des méthodes pour zoomer ou se déplacer (pan) sur la carte. Ces méthodes doivent être implémentées en utilisant les trois événements associés à un contrôle de type "Outil" : MouseDown, MouseMove, MouseUp. On peut utiliser la propriété "Display" sur l'interface IMxApplication ou bien sur l'interface IScreenDisplay. En fait, IAppDisplay hérite de IScreenDisplay, elle possède en plus un certain nombre de propriétés assez utiles non implémentées dans IScreenDisplay (par exemple "count" qui retourne le nombre de ScreenDisplay de l'application).

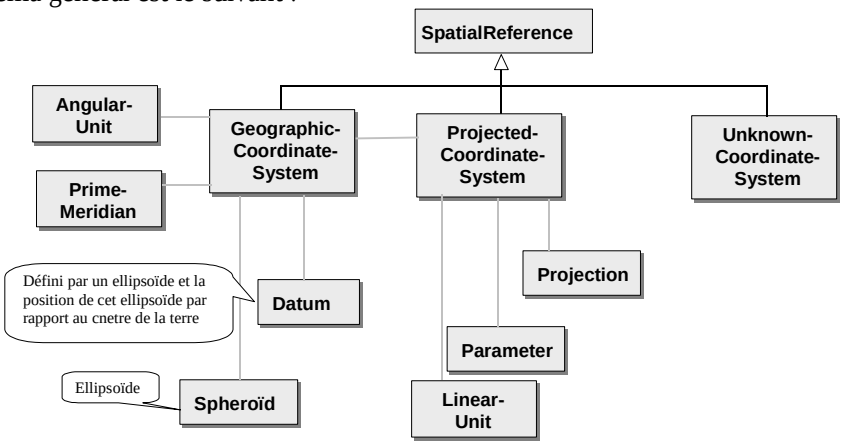
L'exemple VBA suivant montre l'implémentation d'un outil "pan" :

```
Private Sub UIToolControl2_Select()
'On iniitalise les variables globales lorsqu'on sélectionne l'outil
  Set pMxDoc = ThisDocument
  Set pMxApp = Application
  Set pDisp = pMxApp.Display
End Sub

Private Sub UIToolControl2_MouseDown(ByVal button As Long, _
                        ByVal shift As Long, ByVal x As Long, ByVal y As Long)
'Lorsque l'utilisateur appuie sur le bouton gauche de la souris, on récupère
' le point écran sur lequel il a cliqué et on le transforme en coordonnées carte
  Dim pPoint As IPoint
  Set pPoint = pDisp.DisplayTransformation.ToMapPoint(x, y)
  pDisp.PanStart pPoint 'on prépare le display au déplacement à partir de pPoint
  pDisp.TrackPan 'On commence le déplacement interactif du display
End Sub

Private Sub UIToolControl2_MouseMove(ByVal button As Long, _
                        ByVal shift As Long, ByVal x As Long, ByVal y As Long)
'Lorsque l'utilisateur déplace la souris en maintenant le bouton enfoncé,
' on récupère le nouveau point ...
  Dim pPoint As IPoint
  Set pPoint = pDisp.DisplayTransformation.ToMapPoint(x, y)
  pDisp.PanMoveTo pPoint 'et on déplace le display sur cette nouvelle position
End Sub

Private Sub UIToolControl2_MouseUp(ByVal button As Long, _
                        ByVal shift As Long, ByVal x As Long, ByVal y As Long)
'Lorsque l'utilisateur relève le bouton de la souris, on arrête le déplacement
' interactif du display ...
  pDisp.CancelTracker.Cancel
  pDisp.PanStop
  pMxDoc.ActiveView.Refresh '... et on rafraichit la vue
End Sub
```

11.3. REFERENCES SPATIALES ET PROJECTION

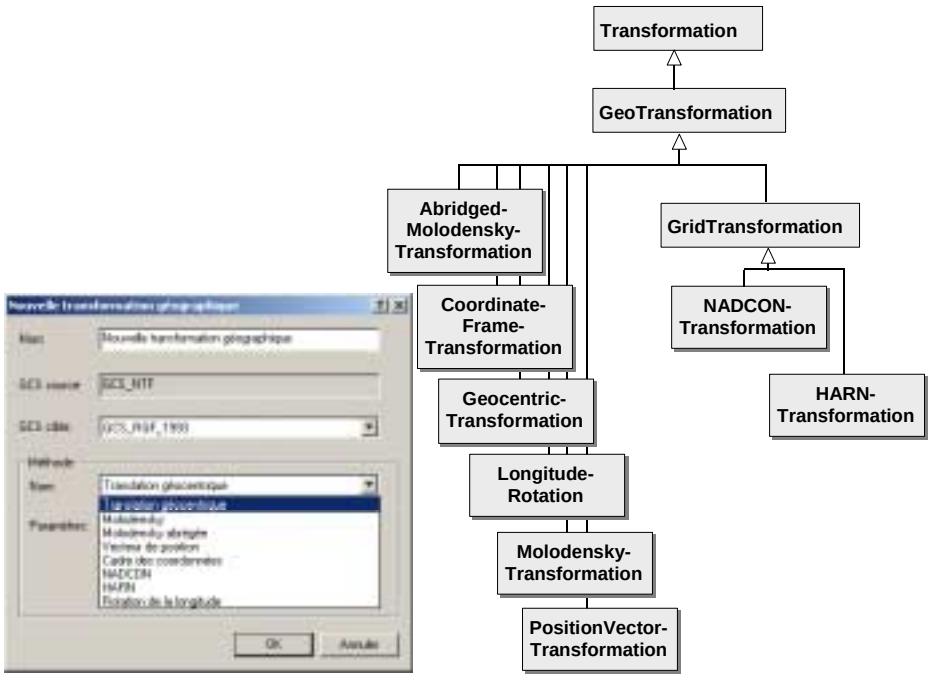
11.3.1. Références spatiales

Le schéma général est le suivant :



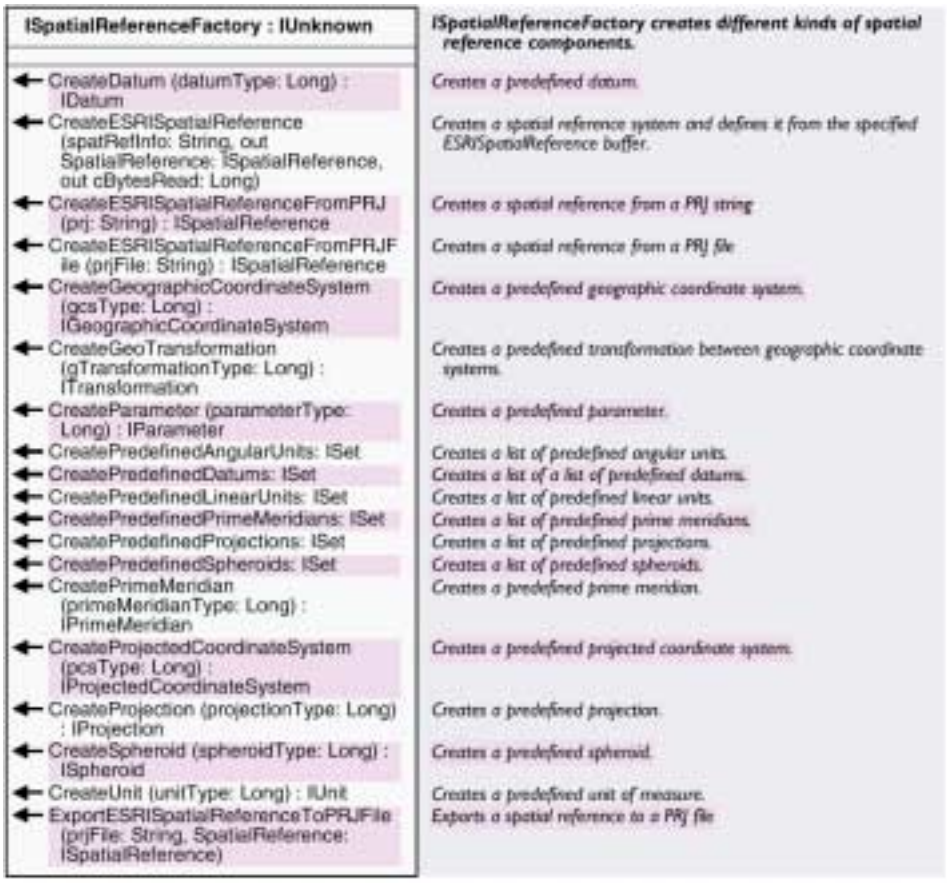
(Extrait de Spatial Reference OMD)

Un système de coordonnées géographiques (système de référence) est défini par un datum, des unités (longitude/latitude en degré ou grades) et un méridien d'origine.
Un système de coordonnées projetées est défini par des unités (mètres ou "feet"), une projection (ie un système d'équations pour passer de coordonnées géographiques en coordonnées planimétriques x,y), des paramètres spécifiques de la projection et un système de coordonées géographiques.
Une transformation géographique (ie un changement de système de référence) est l'opération mathématique permettant de passer d'un système de coordonnées géographiques dans un autre. ArcObjetcs fournit un certain nombre de classes pour effectuer ces transformations, correspondant aux différentes méthodes de passage d'un système dans un autre :



(Extrait de Spatial Reference OMD)

Enfin, la CoClasse "SpatialReferenceEnvironment" implémente l'interface "ISpatialReferenceFactory" qui fournit aux développeurs des méthodes pour "créer" des objets "SpatialReference" prédéfinis :



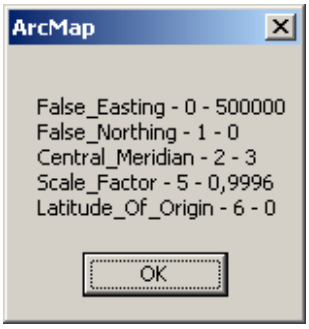
(Extrait de Spatial Reference OMD)

Le code VBA ci-dessous affiche le nom du "SpatialReference" de la première couche d'un document ArcMap. Pour obtenir cette information, il est nécessaire de faire une QI entre IFeatureLayer et IGeodataset afin de pouvoir utiliser la propriété "SpatialReference" :

```
Dim pMxDoc as IMxDocument
Set pMxDoc = ThisDocument
Dim pFLayer as IFeatureLayer
Set pFLayer = pMxDoc.FocusMap.Layer(0)
Dim pGeoDataset As IGeoDataset
Set pGeoDataset = pFLayer 'QI
Dim pSpatialRef As ISpatialReference
Set pSpatialRef = pGeoDataset.SpatialReference
'On affiche le nom et le code de la ref spatiale
MsgBox pSpatialRef.Name & " : " & _
                    pSpatialRef.FactoryCode
```

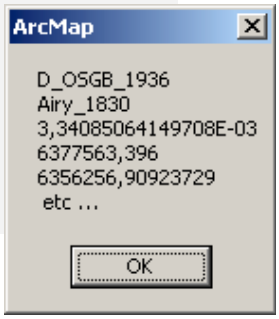
Pour accéder aux paramètres de ce système de projection, il suffit de faire une QI avec IProjectedCoordinateSystem (qui hérite de ISpatialReference). La méthode GetParameters permet d'aller lire ou modifier les valeurs de paramètres de la projection. On crée pour cela un tableau de 16 éléments puis on va les parcourir les uns après les autres.

```
Dim pProjCoordSystem As IProjectedCoordinateSystem
Set pProjCoordSystem = pSpatialRef
Dim pParams(16) As IParameter
pProjCoordSystem.GetParameters pParams(0)
Dim pparam As IParameter
Dim i As Integer
Dim str As String
For i = 0 To 15
  Set pparam = pParams(i)
  If Not pparam Is Nothing Then
    str = str & Chr(13) & pparam.Name & _
          " - " & pparam.Index & _
          " - " & pparam.Value
  End If
Next i
MsgBox str
```



L'exemple suivant montre comment obtenir des informations sur un système de coordonnées géographique existant ("esriSRGeoCS_OSGB1936") en utilisant un ISpatialReferenceFactory :

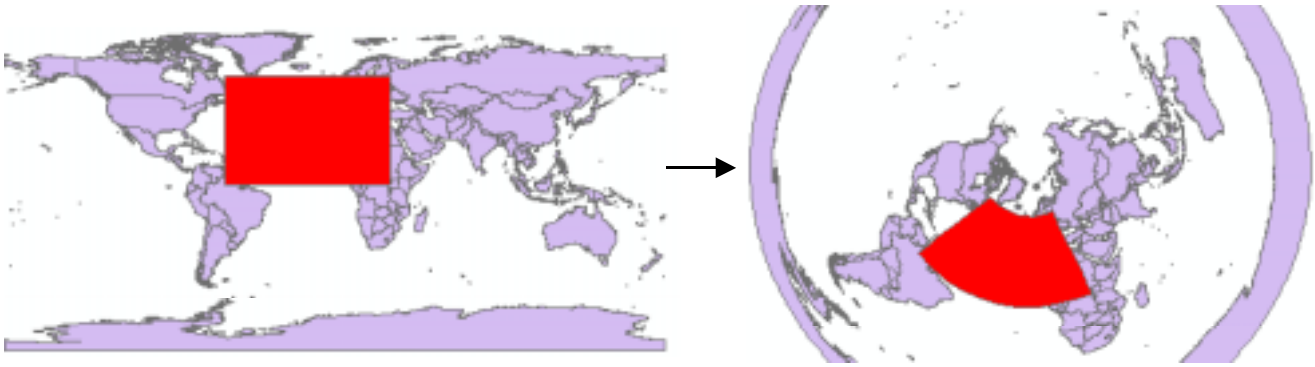
```
Dim pSpRefFact As ISpatialReferenceFactory2
Set pSpRefFact = New SpatialReferenceEnvironment
Set pGeogCoordSystem = _

pSpRefFact.CreateGeographicCoordinateSystem(esriSRGeoCS_OSGB1936)
Dim pDatum As IDatum
Set pDatum = pGeogCoordSystem.Datum
Dim pSpheroid As ISpheroid
Set pSpheroid = pDatum.Spheroid
MsgBox pDatum.Name & Chr(13) & _
       pSpheroid.Name & Chr(13) & _
       pSpheroid.Flattening & Chr(13) & _
       pSpheroid.SemiMajorAxis & Chr(13) & _
       pSpheroid.SemiMinorAxis & Chr(13) & " etc ..."
```



De même, on peut accéder à une projection où une système de référence existant à l'aide d'un fichier .PRJ (méthode "CreateESRISpatialReferencefromPRJFile").

11.3.2. Changement de projection

L'interface "IGeometry" possède une méthode "Project" pour changer les "SpatialReference" d'une géométrie donnée. Pour utiliser cette méthode, il est nécessaire que les références spatiales de la géométrie d'origine soient spécifiées, avant d'appliquer la méthode "Project" vers de nouvelles références spatiales. Il peut également être nécessaire d'appliquer la méthode "Densify" à la géométrie avant de la projeter, afin de rajouter des vertex. Le code VBA ci-dessous projette un rectangle du système de coordonées géographiques "WGS84" vers le systèmes de coordonnées projetées "World Equidistant Conic" :



```
'On suppose pGeometry créé auparavant
Dim pFactory As ISpatialReferenceFactory2
Set pFactory = New SpatialReferenceEnvironment
Dim pGeographic As ISpatialReference
'On affecte la SpatialReference WGS84 à pGeometry
Set pGeographic = pFactory.CreateGeographicCoordinateSystem(esriSRGeoCS_WGS1984)
Set pGeometry.SpatialReference = pGeographic
'On crée une nouvelle SpatialRefercence "World Equidistant Conic"
Dim pProjected As ISpatialReference
Set pProjected = pFactory.CreateProjectedCoordinateSystem(esriSRProjCS_World_EquidistantConic)
'On projette
pGeometry.Project pProjected
```

Si on n'utilise pas la méthode densify sur le rectangle d'origine, on obtient le résultat suivant :

Remarque : pour visualiser la nouvelle géométrie dans la bonne projection, il est nécessaire de modifier également le système de coordonnées du bloc de données (propriété "SpatialReference" sur IMap) :

```
Dim pMap As IMap
Set pMap = pMxDoc.FocusMap
Set pMap.SpatialReference = pProjected
```

12. Analyse spatiale

12.1. SPATIALFILTER

Un "**SpatialFilter**" est un "**QueryFilter**" qui inclut à la fois des critères spatiaux et sémantiques. Il permet de restreindre la "sélection" d'entités à partir d'une couche ou classe d'entité selon des contraintes géométriques.

On utilise l'interface ISpatialFilter pour définir une requête avec un critère géométrique. Trois propriétés doivent être spécifiées pour définir le filtre :

- Geometry : IGeometry => C'est la géométrie de référence utilisée pour filtrer les entités. Par exemple, si on cherche les entités à l'intérieur d'une entité surfacique, c'est le polygone décrivant la localisation de cette entité.
- GeometryField : String => C'est le nom du champ qui porte la géométrie des entités filtrées (la géométrie "cible")
- SpatialRel : esriSpatialRelEnum => C'est une constante correspondant au critère spatial appliqué. Les valeurs sont les suivantes :

  - esriSpatialRelUndefined
  - esriSpatialRelIntersects
  - esriSpatialRelEnvelopeIntersects
  - esriSpatialRelIndexIntersects
  - esriSpatialRelTouches
  - esriSpatialRelOverlaps
  - esriSpatialRelCrosses
  - esriSpatialRelWithin
  - esriSpatialRelContains
  - esriSpatialRelRelation

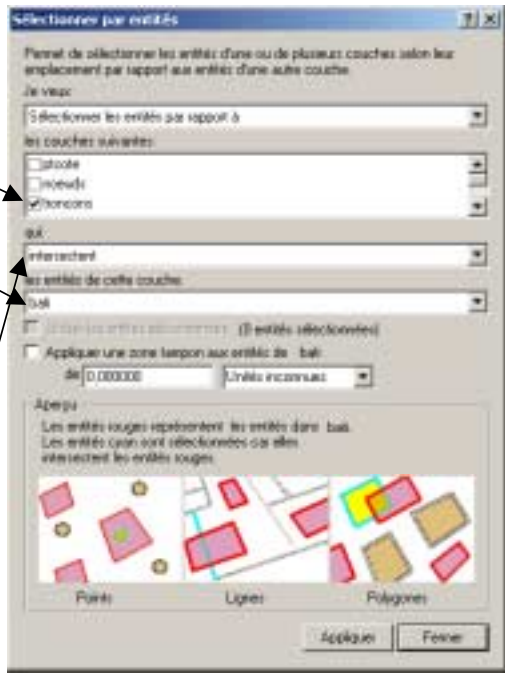
Le code suivant montre simplement la sélection d'entité intersectant une géométrie donnée (pGeometry) :

```
Dim pSpatialFilter as IspatialFilter
Set pSpatialFilter = New SpatialFilter
pSpatialFilter.Geometry = pGeometry
pSpatialFilter.SpatialRel = esriSpatialRelIntersects
Dim pFeatureSelection as IFeatureSelection
Set pFeatureSelection.SelectFeature pSpatialFilter, esriSelectionResultNew, False
```

Il est possible d'utiliser plusieurs SpatialFilter successivement pour exécuter des requêtes plus complexes. ISpatialFilter hérite des propriétés et méthodes de IQueryFilter, on peut donc combiner critères géométriques et sémantiques en utilisant la propriété "WhereClause" sur un filtre spatial.

Pour réaliser une requête par rapport à la géométrie d'un ensemble d'objet (c'est le cas généralement lorsqu'on fait l'équivalent d'un "Sélectionner par entités" sous ArcMap), plusieurs solutions sont possibles. L'une d'entre elle consiste à appliquer le filtre spatial par rapport à chaque objet de référence successivement (dans une boucle avec un curseur par exemple). Une deuxième méthode (beaucoup plus performante) est de construire une seule "géométrie" à partir des géométries de référence et d'appliquer ensuite une seule fois le filtre spatial. Le code VBA suivant utilise cette deuxième méthode pour sélectionner les entités de la couche n°2 intersectant les entités de la couche n°4 :

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pFRouteLayer As IFeatureLayer
Set pFRouteLayer = pMxDoc.FocusMap.Layer(2)
Dim pFeatureSelection As IFeatureSelection
Set pFeatureSelection = pFRouteLayer
Dim pFBatiLayer As IFeatureLayer
Set pFBatiLayer = pMxDoc.FocusMap.Layer(4)

Dim pEnumGeom As IEnumGeometry
Set pEnumGeom = New EnumFeatureGeometry
Dim pEnumGeometryBind As IEnumGeometryBind
'QI entre IEnumGeometryBind et IEnumGeometry
Set pEnumGeometryBind = pEnumGeom
pEnumGeometryBind.BindGeometrySource Nothing, _
     pFBatiLayer.FeatureClass
Dim pGeomFactory As IGeometryFactory
Set pGeomFactory = New GeometryEnvironment
Dim pGeom As IGeometry
'Création d'une seule géométrie à partir de l'ensemble
des entités "bati"
Set pGeom = _
    pGeomFactory.CreateGeometryFromEnumerator(pEnumGeom)

'Création du filtre spatial
Dim pSpatialFilter As ISpatialFilter
Set pSpatialFilter = New SpatialFilter
'.. sur la géométrie pGeom ...
Set pSpatialFilter.Geometry = pGeom
' ..avec la relation "intersectent"
pSpatialFilter.SpatialRel = esriSpatialRelIntersects
' On sélectionne les entités "troncons"
pFeatureSelection.SelectFeatures pSpatialFilter, _
       esriSelectionResultNew, False
pMxDoc.ActiveView.Refresh
```



12.2. SPATIAL OPERATOR INTERFACES

Ces interfaces sont implémentées par la plupart des CoClasses du Geometry OMD et fournissent un panel important d'opérations spatiales (relations topologiques, de proximité ...). Les géométries utilisées lors de ces opérations doivent partager le même système de référence.

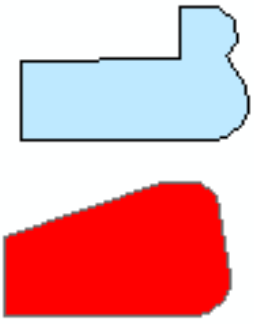
12.2.1. ITopologicalOperator

Les opérations topologiques peuvent être réalisées sur les Polylignes, Polygones, Points et Multipoints. Elles s'appliquent à **une seule "géométrie"** (pas à un jeu de données entier) et permettent par exemple de calculer des **intersections, union, découpages, buffer** ... La liste complète des opérations possibles est donnée par les méthodes de l'interface ITopological :

ITopologicalOperator : IUnknown	Provides access to members that define methods for constructing new geometries based upon topological relationships between existing geometries.
Boundary: IGeometry	Returns the boundary of this geometry.
IsKnownSimple: Boolean	Indicates when this geometry is known to be simple and FALSE if its state is unknown.
IsSimple: Boolean	Indicates when the topological state of this geometry is definitely not simple and TRUE when it is.
TopologyCache (out pTopologyCacheHandle: Long)	Provides a handle to the TopologyCache.
Buffer (Distance: Double) : IGeometry	Constructs a polygon that is the locus of points at a distance less than or equal to a specified distance from this geometry.
Clip (clipperEnvelope: IEnvelope)	Constructs the intersection of this geometry and the specified envelope.
ClipDense (clipperEnvelope: IEnvelope, denseDistance: Double)	Constructs the intersection of this geometry and the specified envelope; densifies lines on window.
ConstructUnion (geometries: IEnumGeometry)	Defines this geometry to be the union of the inputs.
ConvexHull: IGeometry	Constructs the convex hull of this geometry.
Cut (cutter: IPolyline, out leftGeom: IGeometry, out rightGeom: IGeometry)	Splits this geometry into a part left of the specified polyline, and a part right of it.
Difference (other: IGeometry) : IGeometry	Constructs the geometry containing points from this geometry but not the other geometry.
Intersect (other: IGeometry, resultDimension: tagesriGeometryDimension) : IGeometry	Constructs the geometry that is locus of points common to this geometry and the other geometry.
QueryClipped (clipperEnvelope: IEnvelope, clippedGeometry: IGeometry)	Redefines clippedGeometry to be the intersection of this geometry and the clipping envelope.
QueryClippedDense (clipperEnvelope: IEnvelope, denseDistance: Double, clippedGeometry: IGeometry)	Redefines clippedGeometry to be the intersection of this geometry and the clipping envelope; densifies lines on window.
Simplify	Makes this geometry topologically consistent.
SymmetricDifference (other: IGeometry) : IGeometry	Constructs the geometry that contains points from either but not both input geometries.
Union (other: IGeometry) : IGeometry	Constructs the geometry that is the locus of points in one or the other input geometries.

Le code VBA ci-dessous construit un polygone convexe le plus petit possible autour d'une entité sélectionnée :

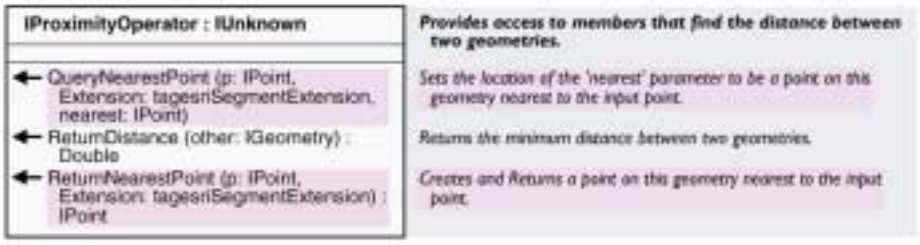
```
Dim pEnumFeat As IEnumFeature
Dim pFeat As IFeature
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Set pEnumFeat = pMxDoc.FocusMap.FeatureSelection
Set pFeat = pEnumFeat.Next

Dim pTopOp As ITopologicalOperator
'On initalise pTopOp avec la gémétrie de l'entité
Set pTopOp = pFeat.Shape
Dim pPolyConvex As IPolygon
'On calcule le polygone convexe de cette géométrie
Set pPolyConvex = pTopOp.ConvexHull
```



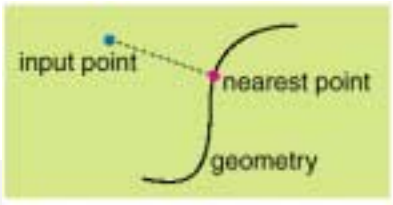
Remarque : La géométrie utilisée pour une opération topologique doit être "simple". Les propriétés "IsKnownSimple" et "IsSimple" indique si cette géométrie est "simple". La méthode "Simplify" transforme la géométrie de façon à ce qu'elle devienne simple.

12.2.2. IProximityOperator

IProximityOperator permet de trouver le point le plus proche d'un point donné sur une géométrie, ou de calculer la distance entre deux géométries existantes. Cette interface est implémentée par les CoClass "Polygon", "Polyline", "Point", "MultiPoint" ainsi que par les sous-classes de "Segment".



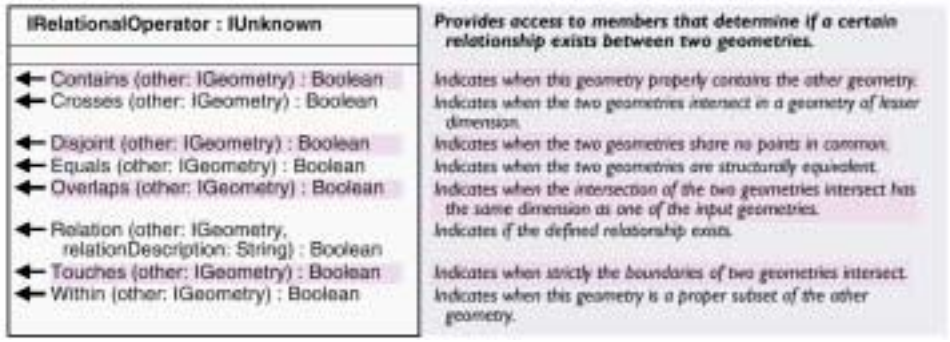
Le code VBA suivant affiche dans un MsgBox les coordonnées du point pNearestPoint le plus proche sur une polyligne pPolyline du point entrée pInputPoint :



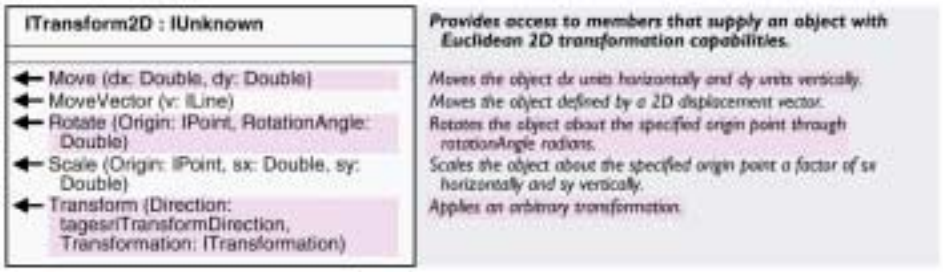

```
Dim pNearestPoint As IPoint
Dim pProxOp As IProximityOperator
Set pProxOp = pPolyLine
Set pNearestPoint = pProxOp.ReturnNearestPoint(pInputPoint, esriNoExtension)
MsgBox "XNearestPoint = " & pNearestPoint.X & ", YNearestPoint = " & pNearestPoint.Y
```

12.2.3. IRelationalOperator

ISpatialOperator fournit des méthodes retournant des booléens pour étudier les relations spatiales entre 2 géométries. Cette interface est implémentée par les CoClass "Polygon", "Polyline", "Point", "MultiPoint" ainsi que par la classe "Envelope". Les méthodes sont les suivantes :



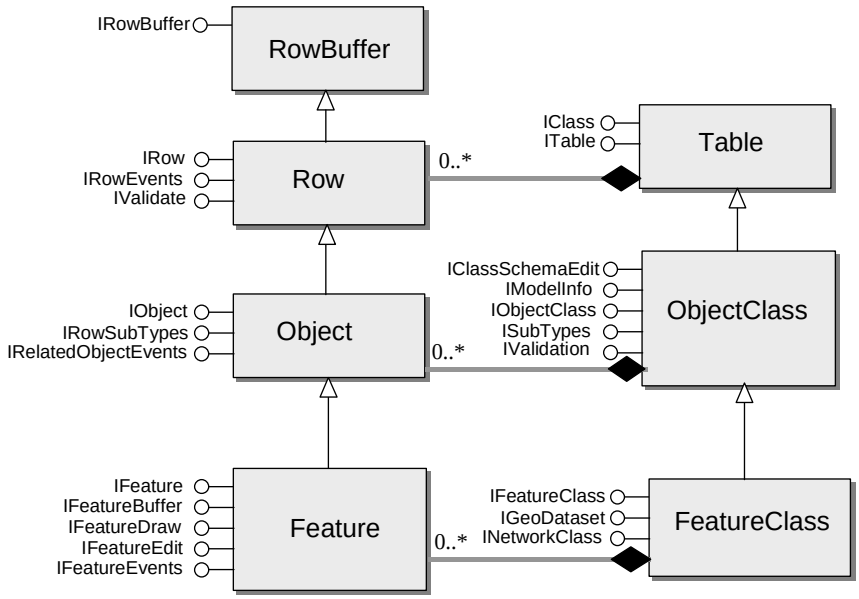
12.3. TRANSFORMATIONS 2D



13. Mise à jour de données

13.1. CREATION D'ENTITES OU AJOUT D'ENREGISTREMENTS DANS UNE TABLE

Comme il a été vu dans le chapitre 5.1, le schéma général décrivant les tables et classes d'entités est le suivant :



Pour créer une nouvelle entité (Feature) dans une classe d'entité (FeatureClass), on utilise la méthode "**CreateFeature**" sur IFeatureClass, qui crée un enregistrement vide. Pour que cette entité soit valide, il faut obligatoirement remplir le champ portant la géométrie ("Shape") avec une géométrie créée au préalable et cohérente avec le type de géométrie de la FeatureClass. Pour ajouter un enregistrement dans une table sans géométrie, on utilise la méthode CreateRow sur l'interface ITable. La méthode "**Store**" sur l'interface "IRow" (dont hérite "IFeature") enregistre les modifications effectuées (créations ou modifications).

Le code VBA ci-dessous montre un exemple d'ajout d'une entité à une classe d'entité donnée (celle associée à la couche 0 de notre document). On suppose que la géométrie pPoint as IPoint a été créée auparavant :

```
Dim pMxDoc as IMxDocument
Set pMxDoc = ThisDocument
Dim pFLayer as IFeatureLayer
Set pFLayer = pMxDoc.FocusMap.Layer(0)
Dim pFClass as IFeatureClass
Set pFClass = pFLayer.FeatureClass
Dim pFeature as IFeature
Set pFeature = pFeatureClass.CreateFeature 'on crée une entité
Set pFeature.Shape = pPoint 'on lui affecte une géométrie
pFeature.Store 'on enregistre
```

Remarque : Les attributs de l'entité autre que la géométrie ne sont pas remplis dans cet exemple

Pour modifier un champ, il suffit d'utiliser la même propriété "Value" que pour lire un champ, puis utiliser la méthode "Store" :

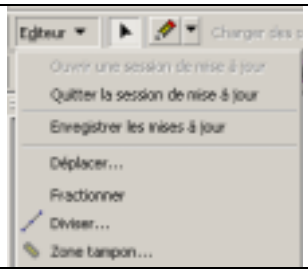
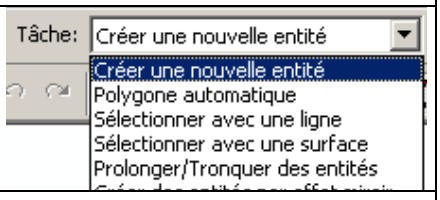
```
pFeature.Value(pFClass.FindField("NOM_RUE_D")) = "RUE PRINCIPALE"
pFeature.Store
```

13.2. UTILISATION D'UNE SESSION D'EDITION

13.2.1. Introduction et rappel des fonctionnalités d'édition

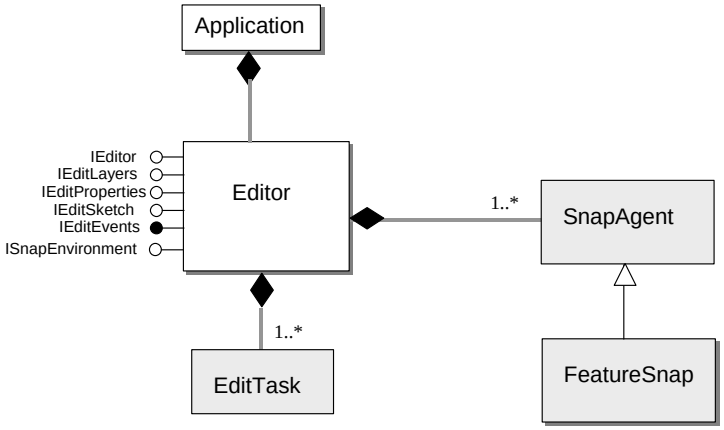
La barre d'outil "Editeur" d'ArcMap est une extension à l'application, fournissant les outils nécessaires à l'édition des données vecteurs géodatabase, shapefile ou couvertures. Les objets ArcObjects décrivant cet "Editeur" permettent au développeur de créer de nouveaux outils, modifier ou supprimer les outils existant ou personnaliser le comportement de l'application.

On peut décomposer les opérations d'édition en quatre types :

- les commandes : l'utilisateur n'a pas besoin de cliquer sur la carte (par exemple, la commande "Zone tampon" du menu "Editeur"). La plupart des commandes sont stockées dans le menu "Editeur".	
- les outils : l'opération nécessite que l'utilisateur clique sur l'écran (par exemple, l'outil "Fractionnement" dans la barre d'outils "Editeur").	
- les tâches de mise à jour : les tâches récupèrent la géométrie stockée dans une "construction" ("sketch") et exécutent une opération spécifique à partir de celle-ci (par exemple "Sélectionner avec une ligne").	
- les événements de mise à jour : ce sont les événements spécifiques pouvant intervenir lors d'une session de mise à jour. Par exemple, on va pouvoir gérer les événements liés à la création de nouvelles entités afin de réaliser des contrôles sur celles-ci.	

L'objet principal du ArcMap Editor OMD est "**l'Editor**", qui implémente un certain nombre d'interfaces correspondant chacune à un groupe de fonctionnalités, entre autres : IEditor, IEditLayers, IEditEvents, IEditProperties, IEditSketch, ISnapEnvironment :

Pour effectuer une mise à jour, l'utilisateur crée ou modifie une "construction" (EditSketch) avec les outils de construction. Lorsque la construction est terminée (FinishSketch), sa géométrie est passée à la tâche courante qui s'exécute sur cette géométrie.



13.2.2. Ouverture et fermeture d'une session d'édition

Pour ouvrir une session d'édition, il faut au préalable récupérer l'extension Editor, soit par son nom, soit par son CLSID (Unique Identifier Object) :

Utilisation de FindExtensionByName :

```
Dim pEditor as IEditor
Set pEditor = Application.FindExtensionByName("ESRI Object Editor")
```

Utilisation de FindExtensionByCLSID :

```
Dim pEditor as IEditor
Dim pID As New esricore.UID
pID = "esricore.Editor"
Set pEditor = Application.FindExtensionByCLSID(pID)
```

L'ouverture et la fermeture d'une session d'édition se font grâce aux méthodes "StartEditing" et "StopEditing" de l'interface "IEditor" et correspondent aux commandes "Ouvrir une session de mise à jour" et "Quitter la session de mise à jour" (Quitter enregistre alors les modifications dans la base de données). "StartEditing" prend comme paramètre le "Workspace" sur lequel on veut effectuer des opérations de mise à jour. "StopEditing" prend en entrée un booléen indiquant si on veut ou non sauvegarder les modifications.

L'exemple VBA suivant montre comment ouvrir une session d'édition sur le workspace correspondant à la première couche vecteur de la carte, en vérifiant au préalable qu'aucune session d'édition n'est déjà ouverte (on suppose qu'on a déjà initialisé pEditor) :

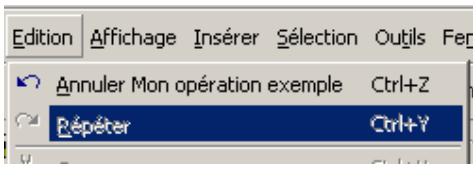
```
Dim pFeatureLayer As IFeatureLayer
Dim pDataset As IDataset
Dim pMap As IMap
Dim pMxDoc As IMxDocument
Dim LayerCount As Integer
Set pMxDoc = Application.Document
Set pMap = pMxDoc.FocusMap
'S'il n'y a pas de session d'édition ouverte
If pEditor.EditState = esriStateNotEditing Then
  'On ouvre une session d'édition sur la première couche vecteur trouvée
  For LayerCount = 0 To pMap.LayerCount - 1
    If TypeOf pMap.Layer(LayerCount) Is IFeatureLayer Then
      Set pFeatureLayer = pMap.Layer(LayerCount)
      Set pDataset = pFeatureLayer.FeatureClass
      pEditor.StartEditing pDataset.Workspace
      Exit For
    End If
  Next LayerCount
End If
```

13.2.3. Opérations de mise à jour et annulations

Les deux méthodes "StartOperation" et "StopOperation" permettent d'ajouter une opération à la pile des "Annuler/Répéter". Elles permettent ainsi de donner un nom à une opération annulable qui apparaîtra dans la commande "Annuler ..." du menu "Edition".

Le code VBA ci-dessous montre l'utilisation de ces deux méthodes pour créeer une opération annulable :

```
pEditor.StartOperation

  'Insérer ici une suite d'instructions
  'réalisant des opération de mise à jour

pEditor.StopOperation "Mon opération exemple"
```

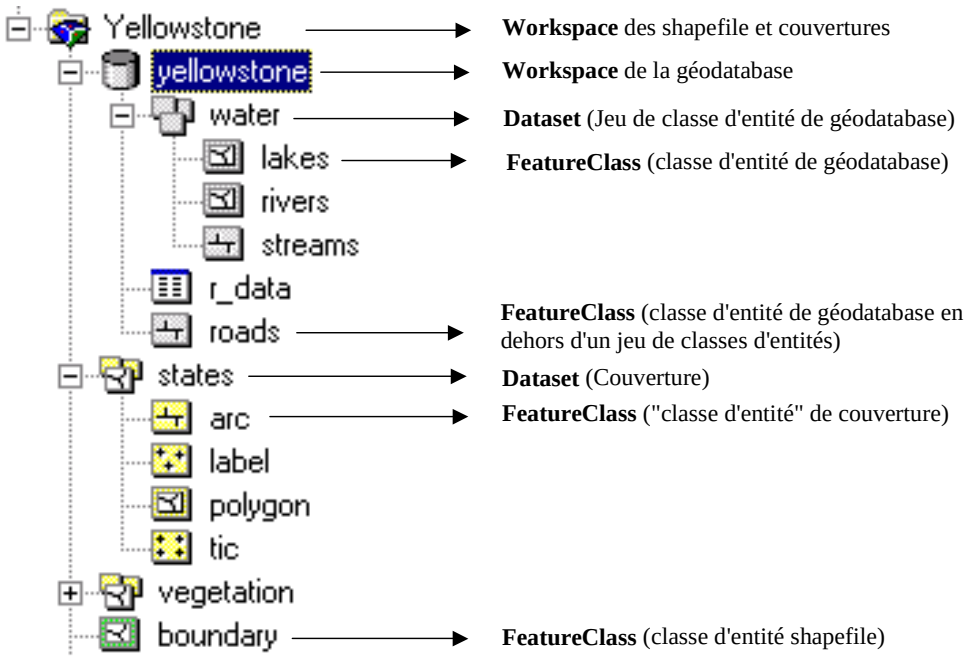


14. Ajout de couche à partir de données existantes

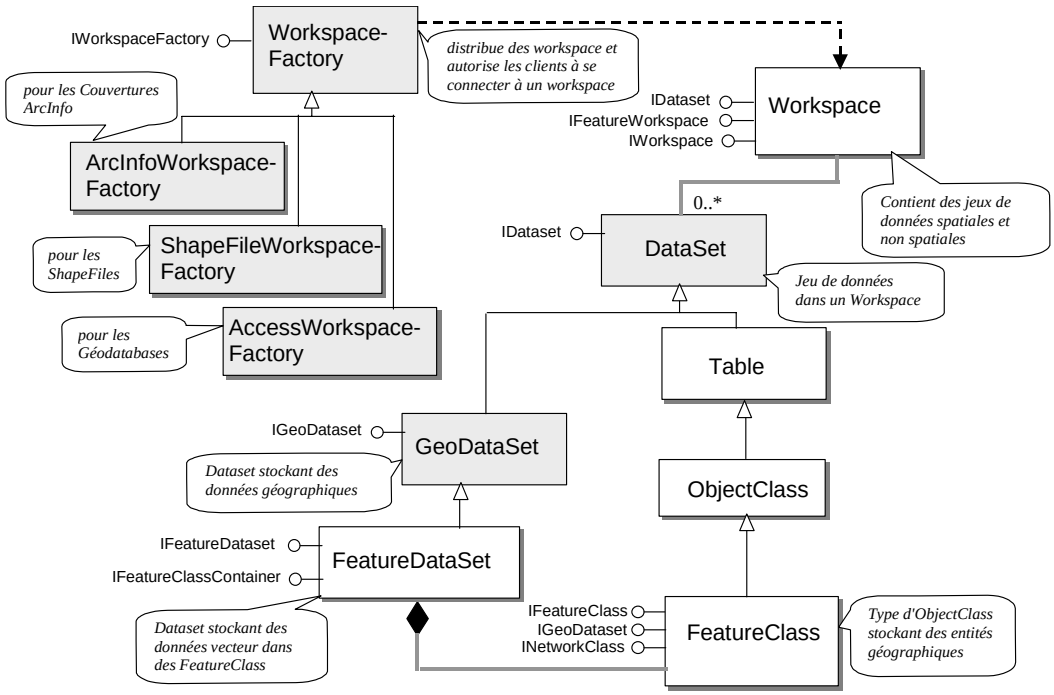
14.1. RAPPEL : LES DIFFERENTS MODES DE STOCKAGE DANS ARCCATALOG

Format / classe dans OMD	Géodatabase	Shapefile	Couverture
Workspace	Fichier Access 2000 (*.mdb)	Répertoire où sont stockés les fichiers de forme	"workspace", ie répertoire où sont stockées les couvertures (il contient toujours un répertoire INFO)
Dataset	Jeu de classes d'entités (FeatureDataSet)	_____	Couverture
FeatureClass	Classe d'entité (FeatureClass)	Fichier de forme (en fait trois fichiers *.shp, *.shx, *.dbf)	Classe d'entité de type fixe : "arc", "polygon", "point", "node", "label" ...

Exemple, par rapport à la visualisation des données sous ArcCatalog :



14.2. SCHEMA GENERAL

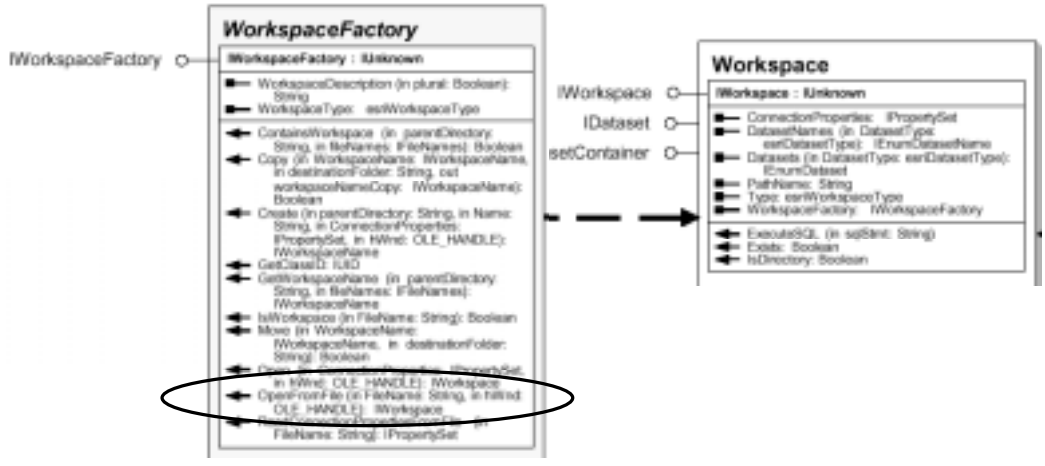


(extrait de Géodatabase OMD)

Pour ajouter une couche à un document ArcMap, il faut dans un premier temps récupérer la classe d'entité (FeatureClass) correspondant à la source des données à ajouter. Pour cela, il faut passer par un "WorkspaceFactory" (une "usine" à Workspace) qui va permettre d'instancier le "Workspace" correspondant à la classe d'entité recherchée. Le type de WorkspaceFactory est différent suivant le format des données (ArcInfoWorkspaceFactory, ShapefileWorkspaceFactory, AccessWorkspaceFactory, mais également SDEWorkspaceFactory, OLEDBWorkspaceFactory etc ...). Toujours suivant le format des données, il faudra ou non ensuite passer par l'intermédiaire d'un FeatureDataset pour accéder à la classe d'entité (ce n'est pas la peine pour les shapefiles ou pour les classes d'entités de géodatabase stockées en dehors d'un jeu de classes d'entités).

14.3. EXEMPLE DETAILLE D'AJOUT DE DONNEES PROVENANT D'UN SHAPEFILE

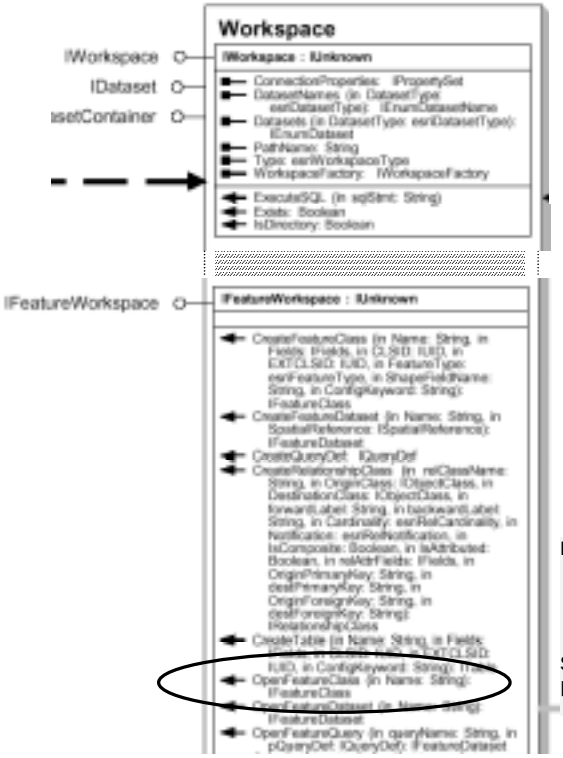
L'exemple suivant ouvre un shapefile sur le disque local et l'ajoute en tant que couche au contenu du document ArcMap.



Récupération du Workspace :

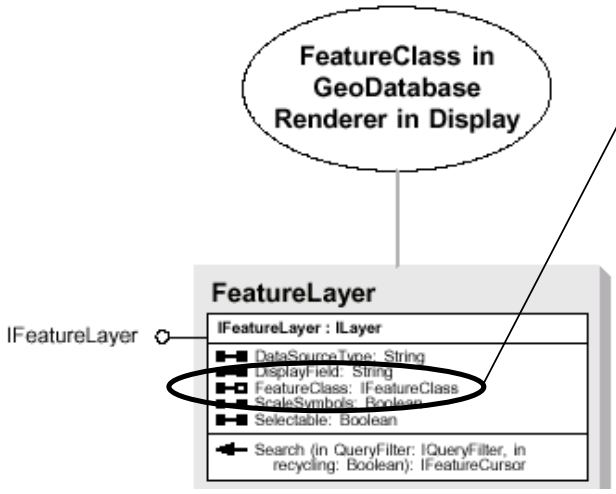
```
'On crée un objet ShapefileWorkspaceFactory pour accéder au workspace qui nous
'intéresse
Dim pWFactory As IWorkspaceFactory
Set pWFactory = New ShapefileWorkspaceFactory
'La méthode OpenFromFile renvoie un objet workspace sur l'interface IWorkspace.
'On fait une QI avec IFeatureWorkspace pour utiliser ensuite la 'méthode
'OpenFeatureClass.
Dim pWorkspace As IFeatureWorkspace
Set pWorkspace = pWFactory.OpenFromFile("D:\users\demo", 0)
'D:\users\demo est le chemin complet du répertoire contenant le shapefile
```

Ouverture de la classe d'entité




```
Dim pFClass As IFeatureClass
'La méthode OpenFeatureClass prend en
'entrée le nom du shapefile sans
'extension *.shp et renvoie un
'IFeatureClass
Set pFClass = _
pWorkspace.OpenFeatureClass("villes")
```

Ajout de la featureClass à la carte ArcMap :



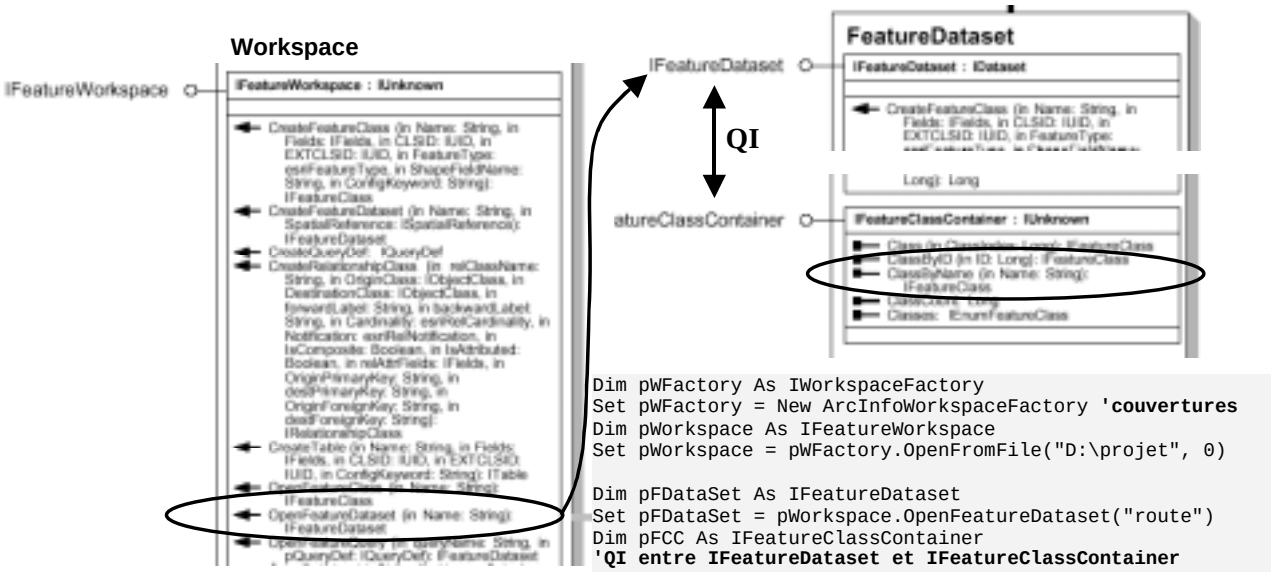
La propriété FeatureClass est en lecture/écriture et permet donc également de définir la source des données pour une nouvelle couche

```
Dim pFLayer As IFeatureLayer
' On crée un nouvel objet FeatureLayer (couche vecteur)
Set pFLayer = New FeatureLayer
' On lui associe la classe d'entité définie précédement
Set pFLayer.FeatureClass = pFClass
' .. et son nom sera celui de la classe d'entité
pFLayer.Name = pFClass.AliasName

Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
'On ajoute enfin cette nouvelle couche à la carte
pMxDoc.AddLayer pFLayer
pMxDoc.ActiveView.Refresh
```

NB : Le principe est le même pour ajouter une classe d'entité de géodatabase stockée en dehors d'un jeu de classes d'entités.

14.4. EXEMPLE D'AJOUT DE DONNEES PROVENANT D'UNE COUVERTURE

Il est nécessaire cette fois de passer par l'intermédiaire d'un "FeatureDataSet" pour accéder à la couverture, avant de pouvoir ouvrir la classe d'entité recherchée. L'exemple ci-dessous ouvre les "arcs" de la couverture "route" stockée dans le workspace "projet" stocké sur le disque "d:\"



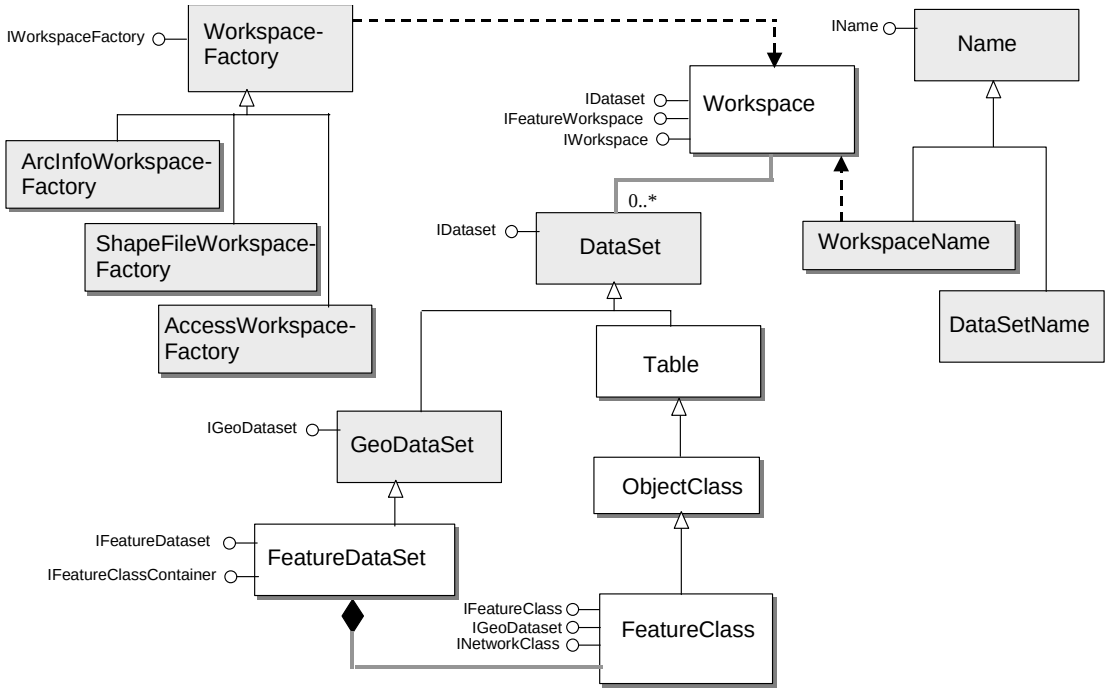

```
Dim pWFactory As IWorkspaceFactory
Set pWFactory = New ArcInfoWorkspaceFactory 'couvertures
Dim pWorkspace As IFeatureWorkspace
Set pWorkspace = pWFactory.OpenFromFile("D:\projet", 0)

Dim pFDataSet As IFeatureDataset
Set pFDataSet = pWorkspace.OpenFeatureDataset("route")
Dim pFCC As IFeatureClassContainer
'QI entre IFeatureDataset et IFeatureClassContainer
Set pFCC = pFDataSet
Dim pFClass As IFeatureClass
'Récupération de la classe par son nom
Set pFClass = pFCC.ClassByName("arc")

Dim pFLayer As IFeatureLayer
Set pFLayer = New FeatureLayer
Set pFLayer.FeatureClass = pFClass
'le nom de la couche sera : route (arc)
pFLayer.Name = pFDataSet.Name & " _
                    (" & pFClass.AliasName & ")"
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
pMxDoc.AddLayer pFLayer
```

NB : Le principe est semblable pour ajouter des classes d'entités de géodatabase stockées dans un jeu de classes d'entités.

15. Création d'une Base de Données

15.1. CREATION D'UNE GEODATABASE

Le schéma général est le suivant (cf. chapitre précédent) :



(extrait de Geodatabase OMD)

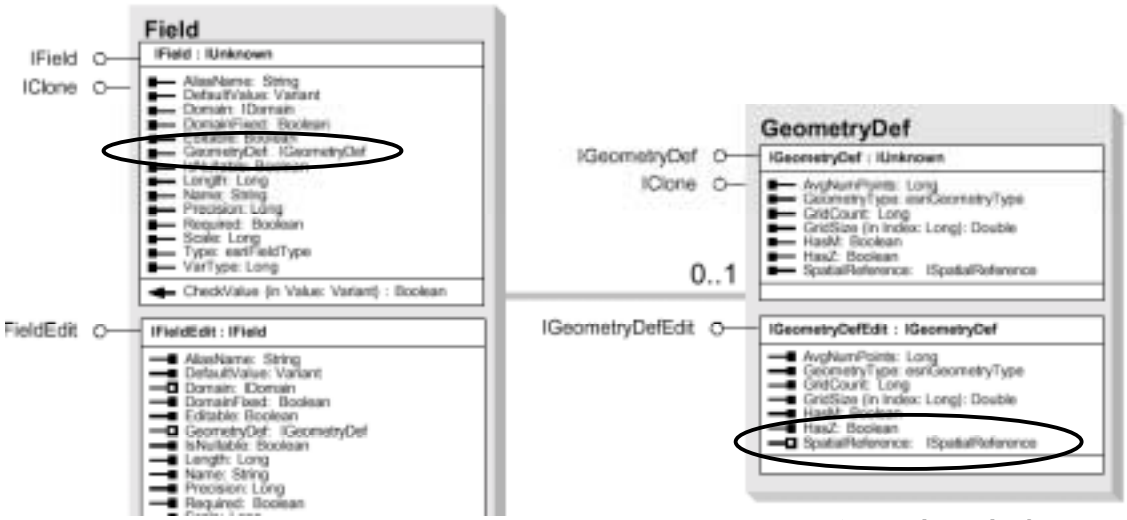
15.1.1. Création d'une géodatabase vide

Tout comme pour ouvrir une géodatabase, on utilise un "WorkspaceFactory". On crée donc un nouvel objet "AccessWorkspaceFactory", puis on utilise la méthode "Create" qui prend en entrée le nom du répertoire de stockage de la géodatabase et le nom de la géodatabase. Cette méthode renvoie un objet "IWorspaceName" qui pemettra ensuite de travailler avec la géodatabase.

```
Dim pAccessWFactory As IWorkspaceFactory
Set pAccessWFactory = New AccessWorkspaceFactory
Dim strMdbFolder As String, strMdb As String
strMdbFolder = "d:\users"
strMdb = "mygdb"
Dim pWorkspaceName As IWorkspaceName
Set pWorkspaceName = pAccessWFactory.Create _
          (strMdbFolder, strMdb, Nothing, 0)
```

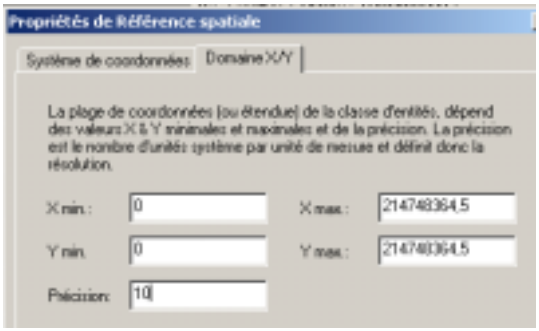
15.1.2. Références spatiales et champ "Géométrie"

Pour créer une classe d'entités, il est nécessaire de créer un champ qui va porter la géométrie (champ "SHAPE" en général). A ce champ (objet "Field") est associé un objet "GeometryDef" qui permet de décrire les paramètres spécifiques à ce type de champ, entre autre la "SpatialReference". Outre la définition du système de coordonnées (cf chap. 11.3), la "SpatialReference" décrit le domaine X/Y et la précision des coordonnées.



(extrait de Geodatabase OMD)

Le code VBA suivant crée un objet "GeometryDef" qui sera utilisé par la suite pour créer une classe d'entité de type "ligne", dans le sytème de coordonnées "Lambert II étendu" et dont le domaine X/Y est le suivant :




```
Dim pGeomDef As IGeometryDefEdit
Set pGeomDef = New GeometryDef
'Création de la SpatialReference à partir d'un fichier .PRJ
Dim pSpatRefFact As ISpatialReferenceFactory
Set pSpatRefFact = New SpatialReferenceEnvironment
Dim pSR As ISpatialReference
Set pSR = pSpatRefFact.CreateESRISpatialReferenceFromPRJFile_
 ("C:\arcgis\arcexe81\Coordinate systems\Projections Françaises\NTF Lambert II étendu.prj")
'Définition du domaine X/Y
pSR.SetFalseOriginAndUnits 0, 0, 10 '(xmin=0, ymin=0, précision de 1/10 de m)
With pGeomDef
  .GeometryType = esriGeometryPolyline '(classe d'entité ligne)
  .GridCount = 1
  .GridSize(0) = 10000
  .AvgNumPoints = 2
  .HasM = False
  .HasZ = False
  Set .SpatialReference = pSR
End With
```

Rappel :

Dans une géodatabase, les coordonnées sont stockées sous forme d'entiers positifs sur 32 bits (ie entre 0 et 2^{31}=2147483647). Pour gérer des coordonnées décimales ou négatives, il peut-être nécessaire d'utiliser un facteur multiplicatif (la "précision") et une translation (xmin et ymin différents de 0). Ces paramètres sont fixés grâce à la méthode

"SetFalseOriginAndUnits", les coordonnées de la carte sont converties par arcGIS comme suit pour être stockées :

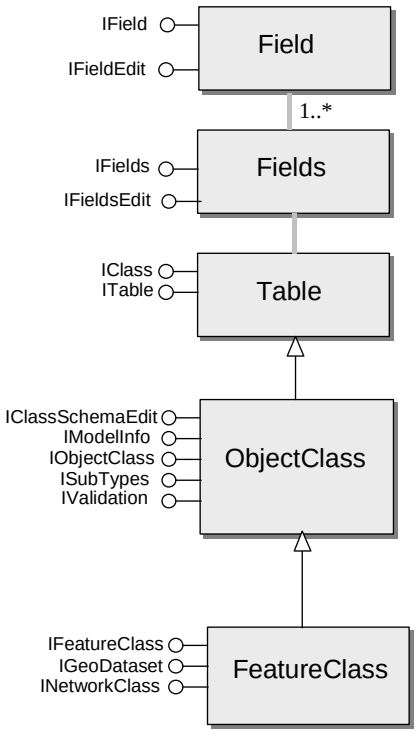
```
ArcGIS X = truncate ((( X coordinate - False X ) * xyunits ) + 0.5 )
ArcGIS X = truncate ((( Ycoordinate - False Y) * xyunits ) + 0.5 )
```

Une autre méthode pour définir le domaine est d'utiliser "SetDomain" qui fixe Xmin, YMin, XMax et YMax, la précision étant alors calculée automatiquement à partir de ces paramètres. Ainsi, pour définir une précision de 1 :

```
pSR.SetDomain 0, 2 ^ 31 , 0, 2 ^ 31
```

15.1.3. Création de champs

Chaque table dans une géodatabase possède une collection ordonnée de champs ("Fields"), avec au moins un champ ("Field"). Chacune de ces deux classes implémente deux interfaces, l'une avec des propriétés en lecture seule, l'autre avec les propriétés équivalentes en écriture seule (IField et IFieldEdit, IFields et IFieldsEdit). Pour créer une nouvelle collection de champs, on utilise IFieldEdit et IFieldsEdit. Suivant le type de champ créé, divers paramètres sont à préciser (longueur pour du texte, GeometryDef pour un champ géométrie, valeur par défaut ...)



L'exemple VBA ci-dessous montre la création de 3 champs "OID", "Shape" et "Nom" respectivement de types "Object ID", "Géométrie" et "Texte". A noter l'utilisation de la méthode cachée(!) "AddField" qui ajoute un champ à une collection de champs :

```
Dim pNomField As IFieldEdit
Set pNomField = New Field
Dim pShapeField As IFieldEdit
Set pShapeField = New Field
Dim pOIDField As IFieldEdit
```

```
Set pOIDField = New Field
pOIDField.Name = "OBJECTID"
pOIDField.Type = esriFieldTypeOID
pNomField.Name = "Nom"
pNomField.Type = esriFieldTypeString
pNomField.Length = 100
pShapeField.Name = "Shape"
pShapeField.Type = esriFieldTypeGeometry
Set pShapeField.GeometryDef = pGeomDef
'Ajout de ces champs à une nouvelle collection de champs
Dim pFieldsEdit As IFieldsEdit
Set pFieldsEdit = New Fields
pFieldsEdit.AddField pOIDField
pFieldsEdit.AddField pShapeField
pFieldsEdit.AddField pNomField
```

15.1.4. CreateFeatureClass

La méthode "CreateFeatureClass" sur IFeatureWorkspace permet de créer une classe d'entité en passant en entrée le nom de la classe d'entité, une collection de champs comportant un champ de type Géométrie, le type d'entité créé (objets simples, réseau, annotations ou personnalisés) et le nom du champ portant la géométrie. Il est indispensable de bien définir l'objet GeometryDef pour que la création soit possible :

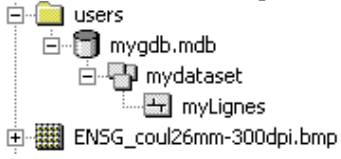
Set *variable* = *object*.**CreateFeatureClass (***Name*, *Fields*, *CLSID*, *EXTCLSID*, *FeatureType*, *ShapeFieldName*, *ConfigKeyword* **)**

Les deux paramètres CLSID et EXTCLSID sont des "Global Unique Identifier" utilisés lorsqu'on souhaite créer des classes d'entités personnalisées.

Pour utiliser "CreateFeatureClass", il faut au préalable ouvrir le workspace. Le code VBA suivant utilise le "WorkspaceName" et les champs créés dans les paragraphes précédents :

```
Dim pName As IName
Set pName = pWorkspaceName 'QI entre IWorkspaceName et IName
Dim pWorkspace As IWorkspace
Set pWorkspace = pName.Open
Dim pFeatureWorkspace As IFeatureWorkspace
Set pFeatureWorkspace = pWorkspace 'QI
Dim pfeatureClass As IFeatureClass
Set pfeatureClass = pFeatureWorkspace.CreateFeatureClass _
  ("myLignes", pFields, Nothing, Nothing, esriFTSimple, "shape", "")
```

La méthode pour créer une classe d'entité dans un jeu de classe d'entité est semblable, la méthode "CreateFeatureClass" s'appliquant à "IFeatureDataSet". A noter toutefois que les références spatiales doivent correspondre (toutes les classes d'entités d'un même jeu de classes d'entités partagent les mêmes références spatiales). Ainsi, par exemple :




```
Dim pFeaturedataSet As IFeatureDataset
Set pFeaturedataSet = pFeatureWorkspace.CreateFeatureDataset _
                      ("mydataset", pSR)
Set pfeatureClass = pFeaturedataSet.CreateFeatureClass _
    ("myLignes", pFieldsEdit, Nothing, Nothing, esriFTSimple, "shape", "")
```

Pour créer une table sans géométrie, on utilise la méthode "CreateTable" qui prend les mêmes arguments que "CreateFeatureClass" en entrée, hormis le "feature type" et le ShapeFieldName".

15.2. EXPORT ET CONVERSIONS

L'exemple suivant montre l'utilisation de la CoClass "ExportOperation" pour exporter le contenu d'une couche ArcMap dans un nouveau fichier de forme (ce qu'on fait dans ArcMap avec un clic-droit sur la couche et le menu "Exporter des données ") :

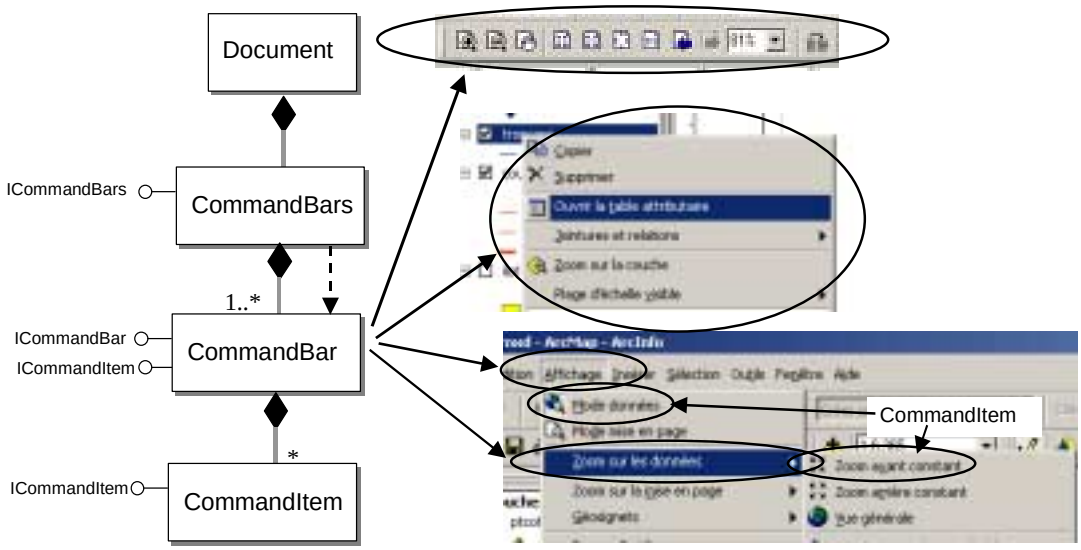

```
'On suppose la couche pFLayer déjà initialisée

' Récupération du DataSetName en entrée
Dim pFClass As IFeatureClass
Set pFClass = pFLayer.FeatureClass
Dim pDataSet As IDataset
Dim pDSName As IDatasetName
Set pDataSet = pFClass
Set pDSName = pDataSet.FullName

'Désignation de la classe d'entité en sortie
Dim pWFactory As IWorkspaceFactory
Set pWFactory = New ShapefileWorkspaceFactory 'export en fichier de forme
Dim pWorkspace As IWorkspace
Set pWorkspace = pWFactory.OpenFromFile("d:\users", 0)
Dim pOutWName As IWorkspaceName
Dim pOutDS As IDataset
Set pOutDS = pWorkspace
Set pOutWName = pOutDS.FullName
Dim pOutFCName As IFeatureClassName
Set pOutFCName = New FeatureClassName
Dim pOutDSName As IDatasetName
Set pOutDSName = pOutFCName
pOutDSName.Name = "export_" & pFClass.AliasName
Set pOutDSName.WorkspaceName = pOutWName

'Export
Dim pExpOp As IExportOperation
Set pExpOp = New ExportOperation
pExpOp.ExportFeatureClass pDSName, Nothing, Nothing, Nothing, _
                          pOutDSName, Application.hWnd
```

Pour davantage de fonctionnalités, utiliser les objets "FeatureDataConverter", qui permettent de programmer des exports du type de ceux présents dans ArcCatalog.

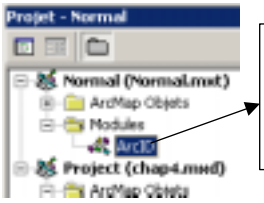
16. Programmation de l'interface utilisateur

16.1. UTILISATION DES COMMANDES EXISTANTES

Il s'agit d'exécuter des commandes existantes en "batch", c'est à dire par programme, sans que l'utilisateur n'ait à intervenir sue les menus ArcMap. Plus généralement, il est possible d'accéder à toute commande, menu ou barre d'outil d'ArcMap via son Unique Identifier Object (**UID**).
Un document ArcMap est composé d'un "CommandBars" qui représente l'ensemble des barres d'outils et menus associés au document en cours. Ce CommandBars est composé de un ou plusieurs "CommandBar" représentant soit une barre d'outil, soit un menu, soit un menu contextuel. Un "CommandBar" est enfin lui-même composé de zéro à plusieurs "CommandItem", un CommandItem étant soit un bouton, un outil ou un item de menu :



Le module "ArcID du projet VBA pour le modèle "Normal.mxt" est un utilitaire qui va permettre, pour un nom de commande passé en entrée, de récupérer l'UID correspondant. On utilise pour cela la méthode "Find" sur ICommandBars" ou "ICommandBar". La liste de l'ensemble des commandes ArcMap et ArcCatalog est fournie dans l'aide en ligne (Technical Documents/ Names and IDs of command and commandBars).
L'exemple ci-dessous ouvre la boîte de dialogue "Imprimer" (Menu Fichier/Imprimer) :



extrait d'ArcID

```
Public Property Get File_Print() As UID
Dim u As New UID
u = "{119591DB-0255-11D2-8D20-080009EE4E51}:7"
Set File_Print = u
End Property
```

extrait de l'aide en ligne :

Type	Caption	Name	Command Category	GUID (CLSID / ProgID)
Command	Print	File_Print	File	{119591D0-0255-11D2-8D20-080009EE4E51} esriCore.MxFileMenuItem

```
Dim cbs As ICommandBars
'La méthode "CommandBars sur IDocument renvoie le commandbars associé au document
Set cbs = ThisDocument.CommandBars
'Execution du CommandItem nommé "File_Print"
cbs.Find(arcid.File_Print).Execute
```

Outre l'exécution de commandes, on peut utiliser les "CommandBars" pour gérer l'affichage des barres d'outils, notament grâce à la méthode "Dock" sur ICommandBar qui va permettre de spécifier la position et l'affichage ou non des barres d'outils. L"exemple ci-dessous affiche la barre de mise en page en dessous de la barre d'outils "Outils" :

```
Dim cbs As ICommandBars
Set cbs = ThisDocument.CommandBars
Dim pToolBar As ICommandBar
Set pToolBar = cbs.Find(arcid.PageLayout_LayoutToolbar)
pToolBar.Dock esriDockBottom, cbs.Find(arcid.Tools_Toolbar)
```

Remarque : il est impossible de cacher le CommandBar correspondant au "menu principal".

16.2. CREATION DE BARRES D'OUTILS, MENUS ET COMMANDES

16.2.1. Création d'une barre d'outil

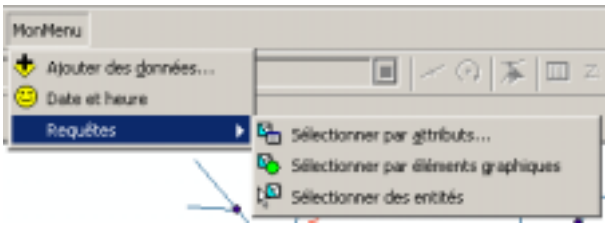
La méthode "Create" sur ICommandBars permet de créer une barre d'outils ou un shortcut menu. Les barres d'outils ainsi créées sont visibles et en position flottante sur la page (esriDockFloat).

Le code VBA ci-dessous crée une barre d'outil "Mes outils" puis ajoute deux commandes ArcMap existantes grâce à la méthode "Add" sur ICommandBar :


```
Sub CreateBar()
  Dim pCBs As ICommandBars
  Set pCBs = ThisDocument.CommandBars
  ' Création de la nouvelle barre d'outils "Mes outils"
  Dim pNewBar As ICommandBar
  Set pNewBar = pCBs.Create("Mes outils", esriCmdBarTypeToolbar)
  'On ajoute des commandes existantes à la barre d'outils
  pNewBar.Add arcid.File_AddData
  pNewBar.Add arcid.PanZoom_FullExtent
End Sub
```

16.2.2. Création de menus et commandes

La méthode "CreateMenu" sur ICommandBar permet de créer des menu et sous-menus dans une barre d'outil ou un menu existant. Ce menu est vide lors de sa création. On peut y ajouter des commandes existantes grâce à la méthode "Add", ou bien créer de nouvelles commandes avec la méthode "CreateMacroItem". Cette méthode permet d'exécuter une procédure écrite par l'utilisateur. L'exemple ci-dessous crée le menu suivant :



```
Public Sub CreateMenu()
  ' Recherche de la barre de menu principal
  Dim pMainMenuBar As ICommandBar
  Set pMainMenuBar = ThisDocument.CommandBars.Find(arcid.MainMenu)
  ' Création du menu "MonMenu" dans la barre de menu principal
  Dim pNewMenu As ICommandBar
  Set pNewMenu = pMainMenuBar.CreateMenu("MonMenu")
  'Ajout d'une commande ArcMap existante
  pNewMenu.Add arcid.File_AddData
  'Création d'une commande exécutant la macro "MaMacro"
  pNewMenu.CreateMacroItem "Date et heure", 1, "Project.Module1.MaMacro"
  Création d'un sous-menu "Requêtes"
  Dim pSousMenu As ICommandBar
  Set pSousMenu = pNewMenu.CreateMenu("Requêtes")
  'Ajout de trois commandes ArcMap existantes
  pSousMenu.Add arcid.Query_AttributeSelect
  pSousMenu.Add arcid.Query_SelectByGraphics
  pSousMenu.Add arcid.Query_SelectFeatures
End Sub

'Macro MyMacro affichant la date et l'heure dans un MsgBox
Public Sub MaMacro()
  MsgBox Now 'Now = fonction VBA renvoyant la date et l'heure
End Sub
```

Remarques :
- les macros appelées par CreateMacroItem doivent être des "Public Subs".
- "CreateMacroItem comprend 3 arguments : le nom de la commande dans le menu, un numéro de pictogramme, le nom complet de la macro VBA (y compris le nom du projet).

16.2.3. Création d'un menu contextuel

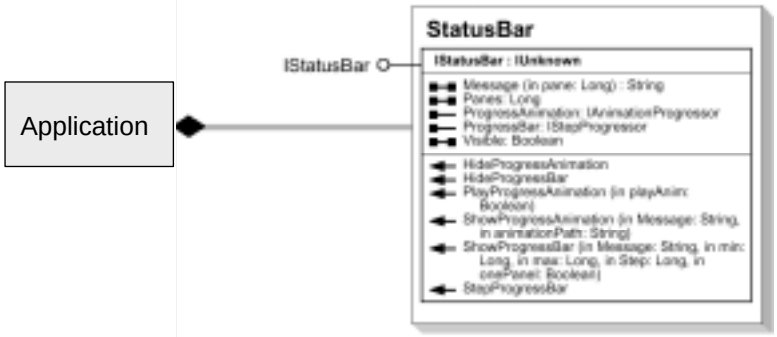
La manipulation des menus contextuels fait appel aux événements pouvant intervenir sur le document (ouverture de document, changement de vue active ...). L'événement "OnContextMenu" s'exécute lorsqu'un utilisateur clique avec le clic-droit sur l'écran (Display).

Le code VBA ci-dessous remplace le menu contextuel qui s'affiche par défaut lorsqu'on clique sur la carte par le menu "Sélection". Le code doit être écrit dans le module "ThisDocument" du projet :

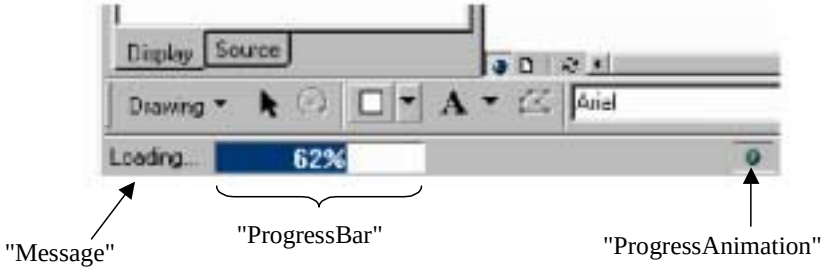
```
Private Function MxDocument_OnContextMenu(ByVal X As Long, _
                                          ByVal Y As Long) As Boolean
  Dim pCBs As ICommandBars
  Set pCBs = ThisDocument.CommandBars
  Dim pMenu As ICommandBar
  'On récupère le menu "Selection_Menu"
  Set pMenu = pCBs.Find(arcid.Selection_Menu)
  'et on l'affiche
  pMenu.Popup
  'Revoie True pour être pris en compte par l'application
  MxDocument_OnContextMenu = True
End Function
```

16.3. BARRE DE PROGRESSION, BARRE D'ETAT

La barre d'état d'ArcMap fournit des informations sur les commandes utilisées, le nombre d'objets sélectionnés etc... Elle peut également montrer l'avancement d'un traitement grâce à une barre de progression. On accède par programme à ce type d'informations grâce à la classe "StatusBar" :



La propriété ProgressBar ainsi que les méthodes HideProgressBar, ShowProgressBar et StepProgressBar permettent de mettre en place une barre de progresion.
La propriété ProgressAnimation ainsi que les méthodes HideProgressAnimation, ShowProgressAnimation et PlayProgressAnimation permettent de "faire tourner la terre" :



L'exemple VBA ci-dessous met en place ces deux éléments dans une boucle de 1 à 900000 :

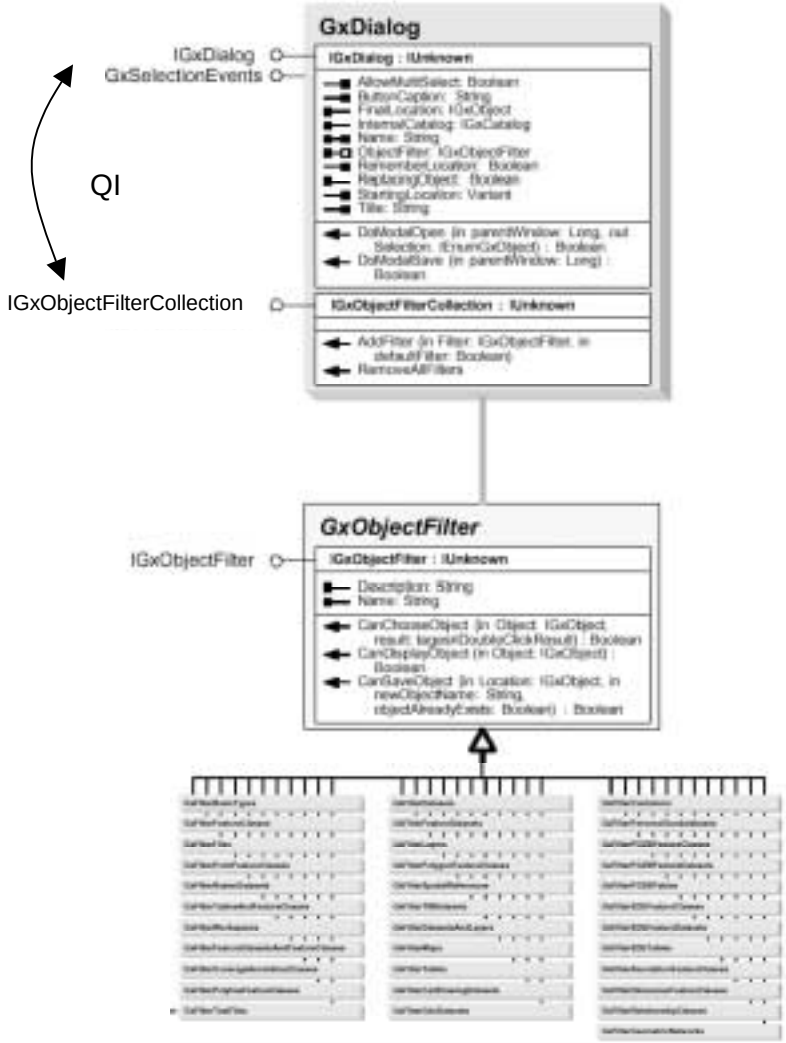
```
Dim pStatusBar As IStatusBar
'On récupère la barre d'état de l'Application en cours
Set pStatusBar = Application.StatusBar
Dim i As Long
'On initialise la ProgressBar
Dim pProgbar As IStepProgressor
Set pProgbar = pStatusBar.ProgressBar
pProgbar.Position = 0
'Et on l'affiche
pStatusBar.ShowProgressBar "Loading...", 0, 900000, 1, True
'On initialise la ProgressAnimation
Dim pProgAnim As IAnimationProgressor
Set pProgAnim = pStatusBar.ProgressAnimation
'On l'affiche et on la fait tourner
pProgAnim.Show
pStatusBar.PlayProgressAnimation True
'Boucle de 1 à 900000 pour simuler un traitement
For i = 0 To 900000
  pStatusBar.StepProgressBar 'on fait avancer la ProgessBar
Next
'On stoppe l'animation et on cache la barre et la terre
pStatusBar.HideProgressBar
pStatusBar.PlayProgressAnimation False
pProgAnim.Hide
```

Remarque : il est également possible de mettre en place des barres de progression dans des boîtes de dialogue en utilisant les objets "ProgressDialog".

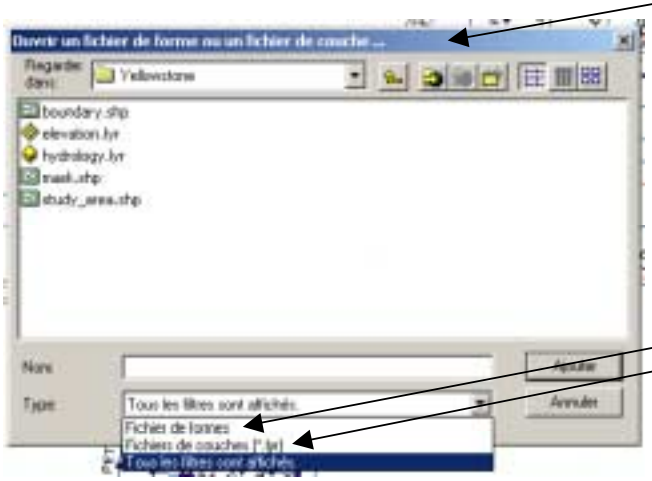
17. Personnalisation ArcCatalog

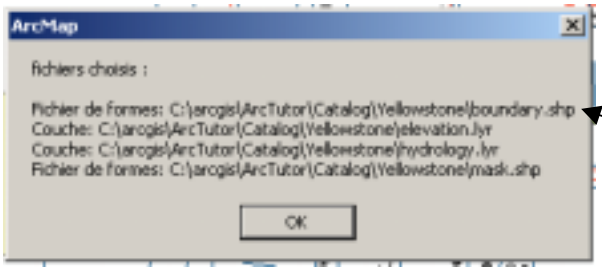
17.1. GxDialog

Les objets "GxDialog" contrôlent les fenêtres de dialogue permettant de lister les données du catalogue. C'est par exemple la fenêtre qu'on obtient en faisant "Ajouter des données" dans ArcMap, ou bien lorsqu'on choisit d'exporter des données d'un fichier de forme vers une couverture dans ArcCatalog. Deux méthodes permettent de créer ces fenêtres : DoModalOpen pour créer une fenêtre du type "Ouvrir" et "DoModalSave" pour créer une fenêtre du type "Enregistrer". Un certains nombre de paramètres sont modifiables : titre, nom du bouton, nombre de fichiers sélectionnables ...
On peut surtout paramétrer le ou les types de fichiers visibles dans la fenêtre grâce au système de GxObjectFilter. Pour permettre à l'utilisateur de ne sélectionner que certains types de fichiers, on va créer une collection de filtres (IGxObjectFilterCollection) que l'on va ensuite appliquer au GxDialog.
La méthode DoModalOpen renvoie un IEnumGxObject qui est une énumération des objets ArcCatalog sélectionnés par l'utilisateur (GxObject). L'abstract classe "GxObject" décrit tous les types d'objets manipulables avec ArcCatalog. Par exemple, la sous-classe "GxPrjFile" porte les fichiers de projection, la sous-classe "GxDataBase" les bases de données reconnues par ArcCatalog etc ...



L'exemple VBA suivant crée une boîte de dialogue de type "Ouvrir" permettant à l'utilisateur de ne sélectionner que les fichiers de forme et les fichiers de couche. Un MsgBox affiche ensuite le type de fichier et son nom complet :

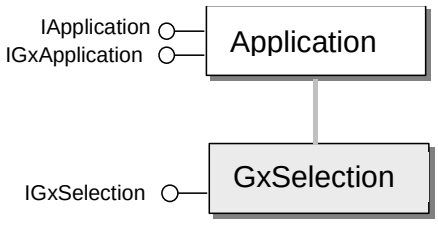





```
Dim pGxDialog As IGxDialog
Set pGxDialog = New gxdialog
'On autorise la sélection de plusieurs fichiers
pGxDialog.AllowMultiSelect = True
pGxDialog.Title = "Ouvrir un fichier de forme _
                   ou un fichier de couche ..."

Dim pShpFilter As IGxObjectFilter
Dim pLyrFilter As IGxObjectFilter
'Création d'un filtre pour les fichiers de forme
Set pShpFilter = New GxFilterShapefiles
'Création d'un filtre pour les fichiers de couche
Set pLyrFilter = New GxFilterLayers

Dim pFilterColl As IGxObjectFilterCollection
'QI entre IGxDialog et IGxObjectFilterCollection
Set pFilterColl = pGxDialog

'On ajoute les filtres à la collection
pFilterColl.AddFilter pShpFilter, True
pFilterColl.AddFilter pLyrFilter, False

Dim pEnumGx As IEnumGxObject
pGxDialog.DoModalOpen ThisDocument.Parent.hWnd, pEnumGx

Dim str As String
pEnumGx.Reset
Dim pGxObj As IGxObject
Set pGxObj = pEnumGx.Next
'Parcours des objets sélectionnés
Do Until (pGxObj Is Nothing)
  str = str & Chr(13) & pGxObj.Category & ": " & _
        pGxObj.FullName
  Set pGxObj = pEnumGx.Next
Loop
MsgBox "fichiers choisis : " & Chr(13) & str
```

17.2. GXAPPLICATION ET GXSELECTION

Dans ArcCatalog, la classe "Application" représente l'application ArcCatalog en cours. Elle permet de gérer l'interface utilisateur (onglets, barres d'outils, menus, ainsi que l'arborescence du catalogue). Elle permet par exemple de travailler sur les objets sélectionnés dans ArcCatalog grâce à la classe "GxSelection" :



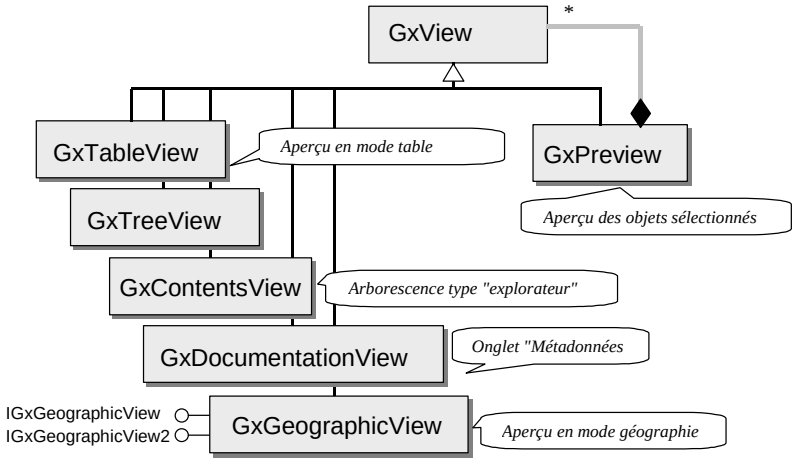
Le code VBA ci-dessous montre comment parcourir et obtenir des informations sur les objets sélectionnés dans ArcCatalog grâce à un IEnumGxObjects :

```
Dim pGxApp As IGxApplication
'QI entre IApplication et IGxApplication
Set pGxApp = Application
'Recupération de la sélection
```

```
Dim pGxSel As IGxSelection
Set pGxSel = pGxApp.selection
Dim pEnumGx As IEnumGxObject
Set pEnumGx = pGxSel.SelectedObjects
'Parcours de la sélection
Dim pGxObj As IGxObject
Dim str As String
Set pGxObj = pEnumGx.Next
Do While Not pGxObj Is Nothing
  str = str & Chr(13) & pGxObj.Name
  Set pGxObj = pEnumGx.Next
Loop
'On affiche le nom des objets sélectionnés
MsgBox str
```

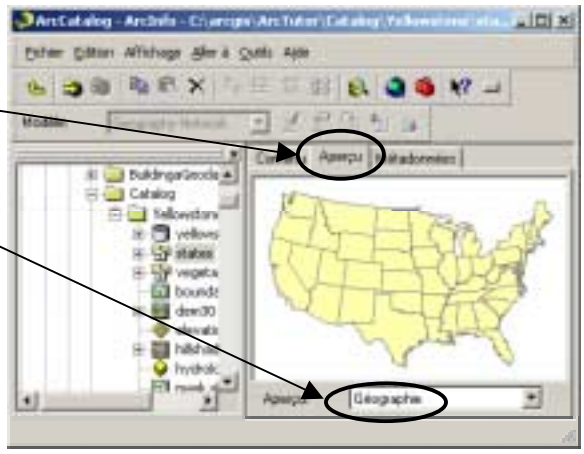
17.3. GXVIEW

GxView est une Abstract Class qui représente toutes les vues possibles d'ArcCatalog. Il existe cinq sous-classes de GxView :



L'exemple VBA suivant montre l'uilisation de GxPreview et GxGeographicView. On vérifie dans un premier temps que l'utilisateur d'ArcCatalog est bien en mode "Aperçu" puis, si c'est le cas, qu'il est bien en mode aperçu "Géographie". Enfin, on récupère l'"ActiveView" et on modifie son extension, afin de dé-zoomer sur l'aperçu :

```
Public Sub ZoomIn()
  Dim pGxApp As IGxApplication
  Set pGxApp = Application 'QI
  'Si on n'est pas dans l'onglet "Aperçu",
  'on sort de la procédure
  If Not TypeOf pGxApp.View Is IGxPreview _
     Then Exit Sub
  Dim pPreview As IGxPreview
  Set pPreview = pGxApp.View
  'Si on n'est pas en mode aperçu "Géographie"
  'on sort de la procédure
  If Not TypeOf pPreview.View Is IGxGeographicView _
     Then Exit Sub
  Dim pGeoView As IGxGeographicView2
  'On récupère alors la "view"
  'et l'activeview correspondante
  Set pGeoView = pPreview.View
  Dim pExtent As IEnvelope
  Set pExtent = pGeoView.ActiveView.Extent
  'On réduit l'extent de façon à dé-zoomer
  pExtent.Expand 0.75, 0.75, True
  pGeoView.ActiveView.Extent = pExtent
  pGeoView.ActiveView.Refresh
End Sub
```



18. index

Bibliographie

- Modeling our World - The ESRI Guide to Geodatabase Design - Michaël Zeiler - ed. ESRI Press
- Programmer ArcInfo avec VBA - Support de cours ESRI France 2000
- Exploring ArcObjects volumes 1 et 2 - ed. Michael Zeiler

Ou trouver de l'aide ?

http://support.esrifrance.fr/
http://arcobjectsonline.esri.com/
liste de diffusion arcsig@georezo.net (inscription : arcsig-subscribe@georezo.net)