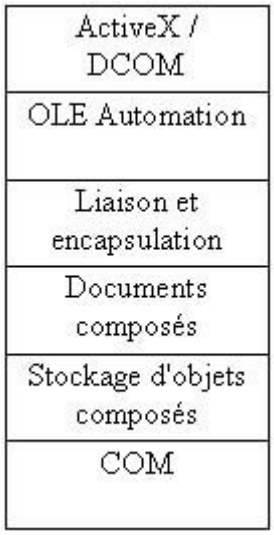
De COM à ActiveX

1. Architecture

Sous l'appellation OLE, on désigne en fait toute l'architecture logicielle de communication inter-applications de Microsoft, allant de la spécification COM aux contrôles ActiveX. OLE c'est en fait un certain nombre de services que le programmeur peut exploiter dans ses applications Windows.

La figure ci-dessous précise les différents services disponibles dans l'environnement OLE.



Cette architecture est basée sur le modèle COM (*Component Object Model*). On trouve ensuite un ensemble de service concernant les documents composés (*Compound Objects*) permettant l'échange de données entre documents. Ces services sont représentés par trois couches dans la figure précédente. Au dessus, on distingue le service *OLE Automation* qui permet à un programme d'exécuter des méthodes d'un objet serveur. Enfin, la dernière brique de cet édifice est ActiveX qui correspond en fait à des contrôles OLE renommés début 96 par Microsoft. On distingue également à ce niveau le modèle *DCOM*, version distribuée du modèle COM.

2. COM

Le *Component Object Model* est, comme nous l'avons dit, le format des applications OLE, leur permettant ainsi de pouvoir communiquer entre elles. Ce format est indépendant du langage de programmation choisi, l'essentiel est que l'exécutable obtenu respecte le modèle. Notons que nous parlons d'exécutable mais en réalité il s'agit plutôt de bibliothèques de fonctions exécutables par plusieurs applications OLE.

Il est important de remarquer qu'OLE est un environnement basé sur des objets client/serveur. Un objet, hébergé par un serveur (bibliothèque *DLL*, application etc.), peut être utilisé par plusieurs programmes clients simultanément.

Du point de vue du programmeur, l'élément de base en programmation OLE est l'*interface*, *classe abstraite* ou *classe virtuelle*. Il s'agit exactement de ce que nous avons vu pour Java : une interface est une classe dont toutes les méthodes sont abstraites. Le programmeur va donc devoir implémenter chacune de ces méthodes pour une interface donnée. L'accès à un objet se fait via des pointeurs sur ses interfaces.

L'interface de base, qu'il est obligatoire d'implémenter pour un objet, est l'interface **IUnknown**. Cette interface est disponible pour tous les objets OLE/COM.

L'interface IUnknown possède, entre autres, une méthode appelée **QueryInterface()** qui permet à un programme utilisant l'objet considéré de prendre connaissance de l'existence d'une interface implémentée dans l'objet. Si l'interface demandée existe, la méthode **QueryInterface()** renvoie un pointeur sur celle-ci afin que l'objet demandeur puisse l'utiliser.

Chaque interface d'un objet est identifiée de façon unique par un *GUID* (*Globally Unique IDentifier*) attribué par Microsoft pour éviter tout conflit d'identifiant. Ce dernier, codé sur 128 bits, est stocké dans la *base des registres* de Windows. Dans cette base, pour chaque GUID, on trouve le nom et la localisation du serveur (DLL par exemple) fournissant l'objet dont on veut accéder à l'une de ses interfaces. On trouve également pour chaque interface un certain nombre d'informations telles que sa version, les types de données reconnus, etc.

A chaque fois que l'interface d'un objet est utilisée par un client, un compteur de référence est incrémenté par ce client, via l'appel d'une fonction **AddRef()**. Lorsque le client n'utilise plus cette interface, il appelle la méthode **Release()** qui décrémente le compteur. Dès que la valeur de ce compteur est nulle, la mémoire utilisée par l'objet considéré est libérée automatiquement.

Une autre interface de base est **IClassFactory**. Cette interface permet de créer un objet, via une méthode appelée **CreateInstance()** à laquelle on indique en argument le GUID de l'objet à instancier. Cette méthode renvoie un pointeur vers l'interface **IUnknown** que nous venons de présenter, permettant ainsi d'utiliser l'objet considéré. Précisons par ailleurs que le mécanisme que nous venons d'évoquer est automatisé par l'appel de la fonction COM **CoGetClassObject()** à laquelle on fournit un GUID qui renvoie donc le pointeur vers l'interface IUnknown correspondante.

Notons enfin qu'il n'existe pas de mécanisme d'héritage pour les interfaces du modèle COM. On utilise à la place soit des *agrégations* (désignation d'un agrégat de plusieurs objets désigné par une seule et même interface IUnknown) soit des *délégations* (où une interface en utilise une autre). Nous n'en dirons pas plus sur ces mécanismes.

3. OLE Automation

OLE Automation est un service très puissant qui permet de manipuler une application en appelant dynamiquement les méthodes des objets, respectant le format COM, qu'elle contient. Cette manipulation est typiquement faite via un langage de scripts tel que *VBA*, mais ce n'est pas une obligation.

En fait, OLE Automation permet à une application de découvrir dynamiquement les objets et méthodes disponibles dans une autre application afin de pouvoir les utiliser. OLE Automation permet également de construire des serveurs d'objets qui sont capables de savoir quelle méthode de quel objet appeler suite à une requête d'une application cliente.

On définit dans OLE Automation des objets automates (*Automation Objects*) qui pourront être manipulés à l'aide de langages de scripts tels que VBA ou être invoqués dynamiquement. Ces applications clientes sont appelées des contrôleurs d'automates.

Nous avons vu plus haut que pour tout objet, on devait obligatoirement implémenter l'interface IUnknown. Dans le cas d'un objet automate, on doit également implémenter de façon obligatoire une autre interface, appelée **IDispatch** et qui permet les invocations dynamiques dont nous venons de parler. Cette interface est utilisée afin d'obtenir le nom de la fonction à appeler tandis que la signature de celle-ci est extraite d'une *librairie de types* des fonctions des objets automates. Cette librairie est généralement stockée dans un DLL ou un fichier annexe.

Notons enfin que la signature de ces fonctions est décrite grâce à un langage appelé *ODL* pour *Object Description Language*.

Passons maintenant à un élément qui nous intéresse tout particulièrement : le contrôle OLE ou *ActiveX*.

4. ActiveX

Les contrôles OLE, renommés ActiveX, sont des composants autonomes permettant de réaliser des applications. Ces composants s'appuient sur COM et OLE Automation que nous venons de voir.

Les contrôles OLE de première génération, appelés *contrôles OCX* (*OLE Control eXtension*), étaient des composants pouvant réagir à des événements extérieurs. La deuxième version de ces contrôles, désormais appelés *contrôles ActiveX* étend les possibilités d'OCX en permettant d'utiliser des composants distribués sur plusieurs systèmes interconnectés par un réseau tel qu'Internet, exploitant le modèle DCOM, extension de COM, que nous décrirons plus loin.

Une application incorporant des contrôles ActiveX est appelée, nous l'avons déjà dit, un *document*. Notons qu'un même document peut contenir des contrôles de types très différents comme des tableaux Excel ou des documents HTML.

Une illustration de la puissance de ce mécanisme est par exemple l'affichage d'un document Word via MSIE. On constate en effet que, dès qu'un tel fichier est récupéré sur un serveur web, il y a incorporation de la barre d'outils de Word *dans* la fenêtre de MSIE, de façon transparente pour l'utilisateur. Ce mécanisme est supérieur à celui que nous avons vu dans le paragraphe précédent qui permettait de manipuler un composant incorporé dans un document, bien qu'un contrôle ActiveX puisse également faire l'objet d'une telle manipulation.

En résumé, un contrôle ActiveX peut être piloté par un document le comprenant, via une interface OLE Automation mais il peut également envoyer des ordres au document le contenant, via un deuxième type d'interface, comme nous venons de l'illustrer.

On distingue par ailleurs le *container ActiveX*, celui qui contient un contrôle, du *component ActiveX* qui est le composant en lui même. Concernant un container, il contient un certain nombre de propriétés de type *ambient* qui sont partagées entre tous les contrôles ActiveX d'un même document. Il peut s'agir par exemple de la couleur ou de la taille de la police de caractères, etc. L'accès à ces données se fait via une interface IDispatch que nous avons présentée plus haut.

Un container doit également pouvoir répondre aux événements générés par un composant ActiveX, également via une interface IDispatch.

Un composant ActiveX quant à lui est caractérisé par un certain nombre de propriétés et d'événements. On distingue quatre types de propriétés :

1. ambiantes (*ambient*),

1. standard,
2. étendues,
3. spécifiques.

Les propriétés ambiantes sont celles que nous avons vues plus haut et sont partagées par tous les composants ActiveX d'un même document. En cas de changement dans ces propriétés, tous les composants sont prévenus par la modification d'un *flag* spécifique

Les propriétés standard sont quant à elles spécifiques à un composant ActiveX et sont similaires aux propriétés ambiantes (couleur, taille des caractères,...).

Les propriétés étendues sont associées à un composant mais ce n'est pas lui qui en a le contrôle. Elles peuvent permettre, par exemple, de spécifier un comportement par défaut dans une boite de dialogue. C'est le document container qui peut les manipuler.

Les propriétés spécifiques sont, comme leur nom l'indique, propres à un composant précis. Elles sont définies par le programmeur du composant.

Les événements, générés par un composant et traités par un container, sont classés en quatre catégories distingues permettant ainsi d'en faciliter l'accès au document :

1. événements de type requête,

1. événements générés *avant* une opération,
2. événements générés *après*,
3. événements générés *pendant* une opération.

Les requêtes sont envoyées par le composant à un document pour lui demander l'exécution d'une certaine action.

Les événements envoyés au document avant et après une opération sont reçus par le document, sans qu'il n'ait à faire une action particulière. On signale juste au document que le composant va débuter et a terminé une certaine opération.

Enfin, les événements produits durant une opération sont en fait des événements standard que l'on rencontre fréquemment dans les interfaces graphiques, à savoir un clic ou déplacement de souris, etc.

Il convient également de préciser que tout composant ActiveX doit être identifié auprès de la base des registres OLE afin qu'il puisse être utilisable. Cette opération est produite automatiquement lorsqu'on récupère un contrôle à partir d'une page web, via la balise HTML **<OBJECT>** dont nous avons parlé brièvement au chapitre précédent. D'ailleurs, ce contrôle est identifié de manière unique par un GUID, dont nous avons également déjà parlé.

Abordons maintenant le dernier service de l'architecture OLE que nous présenterons dans ce chapitre, à savoir l'extension DCOM du modèle COM.

5. DCOM

Distributed COM est la version distribuée du modèle COM qui permet à des objets appartenant à différentes machines de communiquer entre eux via un réseau, alors que COM est limité à une même machine.

DCOM est basé sur l'environnement DCE (*Distributed Computing Environment*) , standardisé par l'OSF, dont il exploite en particulier les appels de procédures distantes RPC (*Remote Procedure Call*), comparables dans leur principe au système RMI dont nous avons parlé pour Java.

Par conséquent, à chaque fois qu'on envoie un message DCOM à un objet distant, ce message est en réalité transporté via RPC.

Dans un environnement DCOM, chaque machine impliquée dispose d'un serveur dit *Object Explorer* qui a pour but de gérer les objets COM disponibles sur la machine considérée. Quand une application

distante fait référence à un objet, l'Object Explorer vérifie que cet objet existe bien dans la base du système sur lequel il tourne. Si c'est le cas, l'accès à cet objet se fait via une évolution de l'interface IUnknown que nous avons présenté plus haut, appelée **IRemUnknown** pour *Remote Unknown*.

Précisons enfin que la gestion des références d'accès aux objets dans un environnement DCOM est quelque peu améliorée pour tenir compte des aléas d'une communication via un réseau. En effet, en cas de plantage d'une machine distante, les références enregistrées par un client ou un serveur risqueraient de ne plus être les mêmes. C'est pourquoi DCOM inclut un mécanisme de *ping* des clients vers les serveurs permettant ainsi d'indiquer à un serveur qu'un client est toujours opérationnel.

Notons par ailleurs que des requêtes ping (vers plusieurs objets d'un même serveur) peuvent être regroupées afin de ne pas sur charger un réseau.

6. OLE en pratique

Une application conteneur est une application qui peut incorporer des éléments liés ou incorporés dans ses documents. Les documents gérés par une application conteneur doivent être capables de stocker et d'afficher des composants de document OLE ainsi que des données créées par l'application elle-même. Une application conteneur doit également permettre aux utilisateurs d'insérer de nouveaux éléments ou de modifier des éléments existants en activant les applications serveur si nécessaire.

Une application serveur ou une application composant est une application qui peut créer des composants de document OLE à utiliser par les applications conteneur. Les applications serveur prennent en charge le glisser-déplacer ou la copie de données dans le Presse-papiers afin que l'application conteneur puisse insérer les données sous la forme d'élément incorporé ou lié. Une application peut être à la fois un conteneur et un serveur.

La plupart des serveurs sont des applications autonomes ou des serveurs complets ; ils peuvent être exécutés en tant qu'applications autonomes ou être exécutés par une application conteneur. Un mini-serveur est un type spécial d'application serveur qui peut être lancé uniquement par un conteneur. Il ne peut pas être exécuté en tant qu'application autonome. Les serveurs Microsoft Draw et Microsoft Graph sont des exemples de mini-serveurs.

Les conteneurs et les serveurs ne communiquent pas directement. Ils communiquent via les DLL système OLE. Ces DLL fournissent des fonctions que les conteneurs et les serveurs appellent et les conteneurs et les serveurs fournissent des fonctions de rappel que les DLL appellent.

En utilisant ce moyen de communication, un conteneur n'a pas besoin de connaître les détails d'implémentation de l'application serveur. Il permet à un conteneur d'accepter les éléments créés par n'importe quel serveur sans avoir à définir les types de serveurs avec lesquels il peut fonctionner. Il en résulte que l'utilisation d'une application conteneur peut tirer parti des applications et des formats de données futurs. Si ces nouvelles applications sont des composants OLE, alors un document composé sera en mesure d'incorporer des éléments créés par ces applications.

7. OLE et VBA, la fonction CreateObject

Crée et renvoie une référence à un objet ActiveX.

Syntaxe
CreateObject(*class,[servername]***)**

La syntaxe de la fonction **CreateObject** comprend les éléments suivants :

Élément	Description
class	**Variant** (**String**). Nom de l'application et classe de l'objet de créer.
servername	Facultatif ; **Variant** (**String**). Nom du serveur réseau où l'objet est créé. Si *servername* est une chaîne vide (""), La machine locale est utilisée.

L'argument *class* utilise la syntaxe *appname.objecttype* et comprend les éléments suivants :

Élément	Description
appname	**Variant** (**String**). Nom de l'application fournissant l'objet.
objecttype	**Variant** (**String**). Type ou classe de l'objet à créer.

Remarques
Chaque application prenant en charge Automation fournit au moins un type d'objet. Par exemple, une application de traitement de texte peut fournir un objet **Application**, un objet **Document** et un objet **Toolbar**.
Pour créer un objet ActiveX, attribuez l'objet renvoyé par la fonction **CreateObject** à une variable objet :

```
' Déclare une variable objet destinée à contenir la
' référence de l'objet. Dim as Object entraîne une
' liaison à l'exécution.
Dim ExcelSheet As Object
Set ExcelSheet = CreateObject("Excel.Sheet")
```

Ce code lance l'application qui crée l'objet, dans le cas présent, une feuille de calcul Microsoft Excel. Une fois l'objet créé, vous y faites référence dans le code à l'aide de la variable objet que vous avez définie. Dans l'exemple suivant, vous accédez aux propriétés et méthodes du nouvel objet à l'aide la variable objet, ExcelSheet, et d'autres objets Microsoft Excel, notamment l'objet Application et la collection Cells.

```
' Rend Excel visible au travers de l'objet Application.
ExcelSheet.Application.Visible = True
' Place du texte dans la première cellule de la feuille.
ExcelSheet.Application.Cells(1, 1).Value = "C'est la colonne A, et la ligne 1"
' Enregistre la feuille dans le répertoire C:\test.xls.
ExcelSheet.SaveAs "C:\ TEST.XLS"
' Ferme Excel en appliquant la méthode Quit sur l'objet Application.
ExcelSheet.Application.Quit
' Supprime la variable objet.
Set ExcelSheet = Nothing
```

Si vous déclarez une variable d'objet avec la locution As Object, une variable contenant une référence à tout type objet est créée. Toutefois, l'accès à l'objet par l'intermédiaire de cette variable est effectué par une liaison tardive, c'est-à-dire que la liaison est créée lors de l'exécution de votre programme. Pour créer une variable objet qui entraîne une liaison précoce, c'est-à-dire une liaison au moment de la compilation du programme, déclarez la

variable objet avec un identificateur de classe spécifique. Par exemple, vous pouvez déclarer et créer les références Microsoft Excel suivantes :

```
Dim xlApp As Excel.Application
Dim xlBook As Excel.Workbook
Dim xlSheet As Excel.WorkSheet
Set xlApp = CreateObject("Excel.Application")
Set xlBook = xlApp.Workbooks.Add
Set xlSheet = xlBook.Worksheets(1)
```

La référence par variable à liaison précoce offre de meilleures performances, mais ne peut contenir qu'une référence à la classe indiquée dans la déclaration.
Vous pouvez transmettre un objet renvoyé par la fonction **CreateObject** à une fonction exigeant un objet en argument. Par exemple, le code suivant crée et transmet une référence à un objet Excel.Application :

```
Call MySub (CreateObject("Excel.Application"))
```

Vous pouvez créer un objet sur un ordinateur réseau distant en transmettant le nom de l'ordinateur à l'argument *servername* de **CreateObject**. Ce nom est identique à une partie du nom de l'ordinateur partagé : pour un nom de partage "\\MonServeur\Public," *servername* est « MonServeur ».

Note Reportez-vous à la documentation COM (voir *Microsoft Developer Network*) pour plus d'informations sur la manière de rendre visible une application à partir d'un ordinateur connecté à distance au réseau. Il sera nécessaire, le cas échéant, d'ajouter une clé de registre pour votre application.
Le code suivant renvoie le numéro de version d'une instance de Excel tournant sur un ordinateur distant appelé MonServeur :

```
Dim xlApp As Object
Set xlApp = CreateObject("Excel.Application", "MonServeur")
Debug.Print xlApp.Version
```

Si le serveur distant n'existe pas ou n'est pas disponible, une erreur au moment de l'exécution se produit.

Remarque Utilisez **CreateObject** lorsqu'il n'existe aucune instance en cours de l'objet. S'il en existe une, une nouvelle instance est lancée et un objet du type indiqué est créé. Pour utiliser l'instance en cours ou pour lancer l'application en chargeant un fichier, utilisez la fonction **GetObject**.

Si un objet a été enregistré comme objet à instance unique, une seule instance de l'objet est créée, quel que soit le nombre d'exécutions de la fonction **CreateObject**.

8.Identificateurs par programmation OLE

Les identificateurs par programmation OLE, ou ProgID (*OLE Programmatic Identifiers*), servent à la création d'objets Automation. Les tableaux ci-dessous répertorient les identificateurs OLE pour les contrôles ActiveX, les applications Microsoft Office et les composants Microsoft Office Web Components.

Contrôles ActiveX

Pour créer les contrôles ActiveX répertoriés dans le tableau ci-dessous, utilisez l'identificateur par programmation OLE correspondant.

Pour créer le contrôle	Utilisez l'identificateur
CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

Microsoft Access
Pour créer les objets Microsoft Access répertoriés dans le tableau ci-dessous, utilisez l'un des identificateurs par programmation OLE correspondants. Si vous utilisez un identificateur sans suffixe de numéro de version, vous créez un objet dans la version de Access la plus récente qui soit disponible sur l'ordinateur sur lequel la macro s'exécute.

Pour créer un objet	Utilisez l'un des identificateurs suivants
Application	Access.Application, Access.Application.9
CurrentData	Access.CodeData, Access.CurrentData
CurrentProject	Access.CodeProject, Access.CurrentProject
DefaultWebOptions	Access.DefaultWebOptions

Microsoft Excel
Pour créer les objets Microsoft Excel répertoriés dans le tableau ci-dessous, utilisez l'un des identificateurs par programmation OLE correspondants. Si vous utilisez un identificateur sans suffixe de numéro de version, vous créez un objet dans la version de Excel la plus récente qui soit disponible sur l'ordinateur sur lequel la macro s'exécute.

Pour créer un objet	Utilisez l'un des identificateurs suivants	Commentaires
Application	Excel.Application, Excel.Application.9	
Workbook	Excel.AddIn	
Workbook	Excel.Chart, Excel.Chart.8	Renvoie un classeur contenant deux feuilles de calcul : l'une pour le graphique, l'autre pour ses données. C'est la feuille du graphique qui est active.
Workbook	Excel.Sheet, Excel.Sheet.8	Renvoie un classeur contenant une feuille de calcul.

Microsoft Graph
Pour créer les objets Microsoft Graph répertoriés dans le tableau ci-dessous, utilisez l'un des identificateurs par programmation OLE correspondants. Si vous utilisez un identificateur sans suffixe de numéro de version, vous créez un objet dans la version de Graph la plus récente qui soit disponible sur l'ordinateur sur lequel la macro s'exécute.

Pour créer un objet	Utilisez l'un des identificateurs suivants
Application	MSGraph.Application, MSGraph.Application.8
Chart	MSGraph.Chart, MSGraph.Chart.8

Composants Microsoft Office Web Components
Pour créer les objets Composants Microsoft Office Web Components répertoriés dans le tableau ci-dessous, utilisez l'un des identificateurs par programmation OLE correspondants. Si vous utilisez un identificateur sans suffixe de numéro de version, vous créez un objet dans la version des Composants Microsoft Office Web Components la plus récente qui soit disponible sur l'ordinateur sur lequel la macro s'exécute.

Pour créer un objet	Utilisez l'un des identificateurs suivants
ChartSpace	OWC.Chart, OWC.Chart.9
DataSourceControl	OWC.DataSourceControl, OWC.DataSourceControl.9
ExpandControl	OWC.ExpandControl, OWC.ExpandControl.9
PivotTable	OWC.PivotTable, OWC.PivotTable.9

RecordNavigationControl	OWC.RecordNavigationControl, OWC.RecordNavigationControl.9
Spreadsheet	OWC.Spreadsheet, OWC.Spreadsheet.9

Microsoft Outlook
Pour créer l'objet Microsoft Outlook indiqué dans le tableau ci-dessous, utilisez l'un des identificateurs par programmation OLE correspondants. Si vous utilisez un identificateur sans suffixe de numéro de version, vous créez un objet dans la version de Outlook la plus récente qui soit disponible sur l'ordinateur sur lequel la macro s'exécute.

Pour créer un objet	Utilisez l'un des identificateurs suivants
Application	Outlook.Application, Outlook.Application.9

Microsoft PowerPoint
Pour créer l'objet Microsoft PowerPoint indiqué dans le tableau ci-dessous, utilisez l'un des identificateurs par programmation OLE correspondants. Si vous utilisez un identificateur sans suffixe de numéro de version, vous créez un objet dans la version de PowerPoint la plus récente qui soit disponible sur l'ordinateur sur lequel la macro s'exécute.

Pour créer un objet	Utilisez l'un des identificateurs suivants
Application	PowerPoint.Application, PowerPoint.Application.9

Microsoft Word
Pour créer les objets Microsoft Word répertoriés dans le tableau ci-dessous, utilisez l'un des identificateurs par programmation OLE correspondants. Si vous utilisez un identificateur sans suffixe de numéro de version, vous créez un objet dans la version de Microsoft Word la plus récente qui soit disponible sur l'ordinateur sur lequel la macro s'exécute.

Pour créer un objet	Utilisez l'un des identificateurs suivants
Application	Word.Application, Word.Application.9
Document	Word.Document, Word.Document.9, Word.Template.8
Global	Word.Global

9. OLE et VBA, la fonction GetObject

Renvoie une référence à un objet fourni par un composant ActiveX.

Syntaxe
GetObject([***pathname***] [**, *class***])**

La syntaxe de la fonction **GetObject** comprend les arguments nommés suivants :

Élément	Description
pathname	Facultatif. Variable de type **Variant** (**String**). Chemin d'accès complet et nom du fichier contenant l'objet à extraire. Si l'argument ***pathname*** est omis, l'argument ***class*** est obligatoire.
class	Facultatif. Variable de type **Variant** (**String**). Chaîne représentant la classe de l'objet.

L'argument ***class*** utilise la syntaxe *appname.objecttype* et comprend les éléments suivants :

Élément	Description
appname	**Variant** (**String**). Nom de l'application qui fournit l'objet.
objecttype	**Variant** (**String**). Type ou classe de l'objet à créer.

Remarques

Utilisez la fonction **GetObject** pour accéder à un objet ActiveX à partir d'un fichier et attribuer cet objet à une variable objet. Utilisez l'instruction **Set** pour attribuer l'objet renvoyé par la fonction **GetObject** à la variable objet. Exemple :

```
Dim CADObject As Object
Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```

Lorsque ce code est exécuté, l'application associée à l'argument ***pathname*** défini est démarrée et l'objet contenu dans le fichier indiqué est activé.
Si l'argument ***pathname*** est une chaîne de longueur nulle (""), la fonction **GetObject** renvoie une nouvelle instance d'objet du type indiqué. Si l'argument ***pathname*** est omis, la fonction **GetObject** renvoie un objet actif du type indiqué. S'il n'existe aucun objet du type indiqué, une erreur se produit.
Certaines applications vous permettent d'activer une partie d'un fichier. Il suffit d'ajouter un point d'exclamation (**!**) à la fin du nom de fichier et d'indiquer à la suite une chaîne identifiant la partie du fichier à activer. Pour plus d'informations sur la création de cette chaîne, consultez la documentation relative à l'application utilisée pour créer l'objet.
Dans une application de dessin, par exemple, vous pouvez avoir un dessin constitué de plusieurs plans dans un fichier. Le code suivant vous permet d'activer un plan d'un dessin nommé SCHEMA.CAD :

```
Set LayerObject = GetObject("C:\CAD\SCHEMA.CAD!Layer3")
```

Si vous n'indiquez pas l'argument ***class***, Automation se base sur le nom de fichier que vous fournissez pour déterminer l'application à démarrer et l'objet à activer. Certains fichiers peuvent toutefois gérer plusieurs classes d'objets. Ainsi, un dessin peut prendre en charge trois types d'objets différents : un objet **Application**, un objet **Drawing** et un objet **Toolbar**, appartenant tous au même fichier. Pour indiquer l'objet à activer parmi ceux du fichier, utilisez l'argument facultatif ***class***. Exemple :

```
Dim MyObject As Object
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW", "FIGMENT.DRAWING")
```

Dans l'exemple, FIGMENT est le nom d'une application de dessin et DRAWING est l'un des types d'objets pris en charge par cette application.
Une fois qu'un objet est activé, vous pouvez y faire référence dans le code en utilisant la variable objet que vous avez définie. Dans l'exemple précédent, vous accédez aux propriétés et aux méthodes du nouvel objet à l'aide de la variable objet MyObject. Exemple :

```
MyObject.Line 9, 90
MyObject.InsertText 9, 100, "Bonjour à tous."
MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

Note Utilisez la fonction **GetObject** lorsqu'il existe une instance en cours de l'objet ou si vous souhaitez créer l'objet avec un fichier déjà chargé. S'il n'existe aucune instance en cours et si vous ne voulez pas démarrer l'objet en chargeant un fichier, utilisez la fonction **CreateObject**.

Si un objet est enregistré comme objet à instance unique, une seule instance de l'objet est créée quel que soit le nombre d'exécutions de la fonction **CreateObject**. Dans le cas d'un objet à instance unique, la fonction **GetObject** renvoie toujours la même instance lorsqu'elle est appelée avec la syntaxe à chaîne de longueur nulle ("") et provoque une erreur si l'argument ***pathname*** est omis. Vous ne pouvez utiliser la fonction **GetObject** pour obtenir une référence à une classe créée à l'aide de Visual Basic.

10. Exemple d'utilisation de GetObject avec un document Excel

Cet exemple utilise la fonction **GetObject** pour obtenir une référence à une feuille de calcul Microsoft Excel spécifique (MyXL). Il utilise la propriété **Application** de la feuille de calcul pour rendre Microsoft Excel visible, pour fermer l'application, etc. Le premier appel à la fonction **GetObject** entraîne une erreur si Microsoft Excel n'est pas déjà en exécution. Dans notre exemple, l'erreur a pour conséquence d'attribuer la valeur True à l'indicateur ExcelWasNotRunning. Le deuxième appel à la fonction **GetObject** indique le fichier à ouvrir. Si Microsoft Excel n'est pas déjà en exécution, le deuxième appel lance l'application et renvoie une référence à la feuille de calcul représentée par le fichier indiqué, montest.xls. Ce fichier doit se trouver à l'emplacement spécifié ; dans le cas contraire, l'erreur Erreur Automation Visual Basic est générée. L'exemple de code rend ensuite Microsoft Excel et la fenêtre contenant la feuille de calcul indiquée visibles. Enfin, si Microsoft Excel n'était pas en exécution précédemment, le code utilise la méthode **Quit** de l'objet **Application** pour fermer Microsoft Excel. Si l'application était déjà en exécution, le code ne tente pas de la fermer. La référence elle-même est libérée en recevant la valeur **Nothing**.

```
' Déclare les routines d'API nécessaires:

Declare Function FindWindow Lib "user32" Alias _
"FindWindowA" (ByVal lpClassName as String, _
            ByVal lpWindowName As Long) As Long

Declare Function SendMessage Lib "user32" Alias _
"SendMessageA" (ByVal hWnd as Long,ByVal wMsg as Long, _
            ByVal wParam as Long, _
            ByVal lParam As Long) As Long

Sub GetExcel()
Dim MyXL As Object    ' Variable devant contenir la
                  ' référence à Microsoft Excel.
```

```
Dim ExcelWasNotRunning As Boolean    ' Indicateur de libération finale.

' Test pour déterminer si une copie de Microsoft Excel
' est déjà en exécution.
On Error Resume Next    ' Retarde la récupération d'erreur.
' La fonction Getobject appelée sans le premier
' argument renvoie une référence à une instance de
' l'application. Si l'application n'est pas en
' exécution, une erreur se produit.
Set MyXL = Getobject(, "Excel.Application")
If Err.Number <> 0 Then ExcelWasNotRunning = True
Err.Clear     ' Efface l'objet Err si une erreur s'est produite.

' Vérifie si Microsoft Excel est en exécution.
' Dans ce cas, l'ajoute à la table Running Object.
DetectExcel

' Définit la variable objet faisant référence au fichier à ouvrir.
Set MyXL = Getobject("c:\vb5\MONTEST.XLS")

' Affiche Microsoft Excel par l'intermédiaire de sa
' propriété Application. Affiche ensuite la fenêtre
' contenant le fichier à l'aide de la collection
' Windows de la référence à l'objet MyXL.
MyXL.Application.Visible = True
MyXL.Parent.Windows(1).Visible = True
    ' Effectue des opérations sur votre
     ' fichier ici.
    ' ...
' Si cette copie de Microsoft Excel n'était pas en cours d'exécution lorsque vous avez commencé, fermez-la
' à l'aide de la méthode Quit de la propriété Application.
' Notez que si vous tentez de quitter Microsoft Excel,
' la barre de titre clignote et un message s'affiche
' vous demandant si vous souhaitez enregistrer les
' fichiers chargés.
If ExcelWasNotRunning = True Then
    MyXL.Application.Quit
End IF

Set MyXL = Nothing  ' Libère la référence à l'application
                    ' et à la feuille de calcul.
End Sub

Sub DetectExcel()
' La procédure détecte une instance d'Excel en
' exécution et l'inscrit.
    Const WM_USER = 1024
    Dim hWnd As Long
' Si Excel est en exécution, cet appel d'API renvoie
' son descripteur.
    hWnd = FindWindow("XLMAIN", 0)
    If hWnd = 0 Then    ' 0 signifie qu'Excel n'était
                        ' pas en exécution.
        Exit Sub
    Else
    ' Excel est en exécution, donc utilise la fonction d'API SendMessage pour l'entrer dans la table
    ' Running Object.
        SendMessage hWnd, WM_USER + 18, 0, 0
    End If
End Sub
```