Conférence DL190 / Composants : RichEdit OLE

Cette session montre l ’extension du composant RichEdit par le support de l ’interface COM permettant l ’insertion d ’objets OLE, y compris dans le TDBRichEdit. Elle démontre notamment l ’implémentation du stockage d ’objets COM et des Call-Back OLE, fréquents dans les API Windows avancées.

Catégorie : Composants

niveau : Avancé

Pré Requis : Très bonne connaissance de la VCL, de la création de composants et du développement sous Delphi.

Agenda

- Le Modèle COM et OLE
- Les MFC
- Le Contrôle MFC cRichEditCtrl
- L ’Interface Rich Edit OLE
- Un composant RichEdit intégrant graphismes et objets OLE pour Delphi et C++ Builder.

Tout développeur s ’est vite retrouvé frustré devant le composant TRichEdit de Delphi : il ne gère que du texte… alors qu ’un simple utilitaire comme WordPad sait intégrer des graphismes et des objets OLE !

J ’ai choisi ici de vous proposer la réalisation de ces fonctions qui font la différence entre un logiciel amateur et un autre, plus .. « pro ».

Si la finalité n ’était que de vous fournir un nouveau composant Delphi, je vous aurais remis une disquette et nous aurions tous pu aller vaquer à d ’autres occupations…

En fait il ne s ’agit que d ’un prétexte pour lever le voile sur un monde fantastique qui se trouve là, tout prêt de vous, et que vous ne soupçonnez peut-être pas : celui de COM (DCOM) et d ’OLE.

La VCL de Delphi n ’est qu ’une façade pour celui qui sait voir plus loin que les propriétés publiées dans l ’inspecteur d ’objets. Comme dans les meilleurs Hercules Poirot et autres Sherlock Holmes il existe des portes dérobées qui ne s ’ouvrent que si on appui sur le chandelier de gauche au dessus de la cheminée…

La bibliothèque bascule… allumez vos torches et suivez-moi...

Le Modèle COM :

Component Object Model

- Architecture logiciel permettant la construction d'applications depuis des composants binaires.
- COM est la fondation d'un édifice de services logiciels de haut niveau (par ex. OLE)
- Les services OLE couvrent de nombreuses fonctions système (documents composites, contrôles personnalisés, transfert de données…)

Notre voyage commence par la découverte des fondations : COM.

Modèle Objet de Composant : une architecture qui fournit l'assise à tout l'édifice mis en place depuis Windows 95.

Construit sur cette base se trouvent les services OLE.

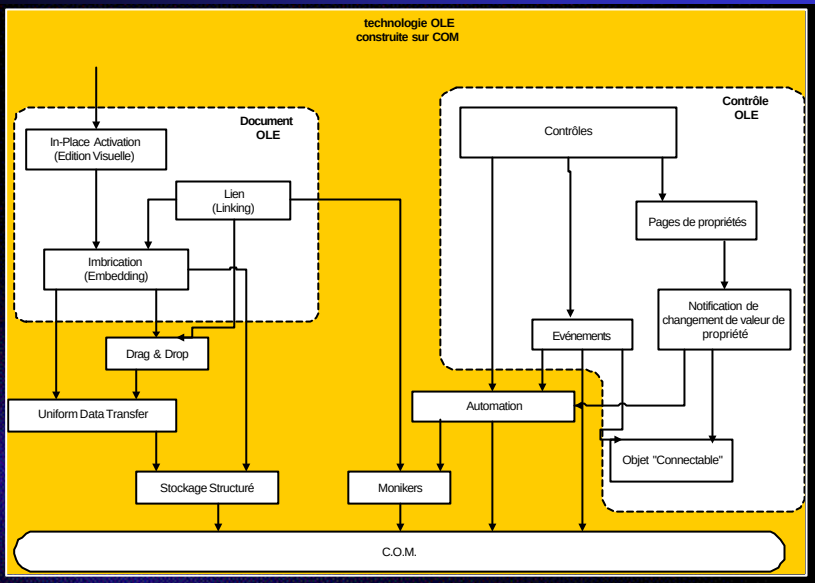
Si OLE était au départ un sigle signifiant « Objets liés et imbriqués » qui définissait même un format de fichier composite il faut se rendre à l'évidence que ce sigle a totalement perdu cette signification. MS n'en donne d'ailleurs plus la traduction du sigle. OLE est aujourd'hui un concept de services reposant sur COM.

Ces services de haut niveau sont exploitables depuis les applications Windows. On trouve de tout : documents composites, contrôles personnalisés, scripting inter-application et autres transferts de données.

Tous ces services partagent un mécanisme commun autorisant l'utilisation et la connexion de composants binaires dérivés de toute combinaison de composants existants venant de fournisseurs différents, écrits dans des langages différents.

Ce mécanisme est fourni par COM.

L ’architecture OLE



On voit ici, entourés par des pointillés, deux grandes zones :

* Les documents OLE

* Les contrôles OLE

Les premiers supportent l ’activation sur site (in-place), le lien (linking) et l ’imbrication (embedding) ce qui se rapproche assez fidèlement de la première définition de OLE.

Les seconds sont des classes d ’objets (services ou GUI) supportant des pages de propriétés, des notifications et des événements (ce qui ressemble à ce que la VCL propose).

Ces deux constructions reposent sur COM qui fournit les mécanismes de base.

L ’architecture COM

- Définit un standard binaire de composants
- Est indépendante du langage
- Est multi plate-forme
- Autorise des évolutions d ’app. robustes
- Supporte les extensions de compo.
- Propose un même mécanisme pour dialoguer en local ou distant
- Autorise le partage de mémoire
- Offre un support étendu des erreurs
- Autorise le dé/chargement dynamique des compo.

* COM est un standard binaire pour l ’interopérabilité des composants. On remarquera que ceci est l ’inverse de toutes les démarches qui ont échouées bien plus sensées : COM généralise à partir du binaire et non du haut niveau (langage).

* COM est indépendant du langage du moment que ce dernier peut générer du binaire.

* COM est disponible sur plusieurs plates-formes : Windows 3, Windows 95, NT, Apple Macintosh, divers UNIX, LINUX.

* COM autorise le partage de mémoire entre composants.

* COM offre un modèle unique pour la communication des composants que cela soit au sein d ’un même process ou bien de process différents et même de process se trouvant reliés par un ou plusieurs réseaux.

Il faut noter que COM est une architecture générale. MS l ’utilise dans certains cadres (contrôles, documents composites…) mais chaque développeur peut en tirer partie pour ses propres besoins.

COM : Unificateur universel ?

Comment offrir l 'Interopérabilité ?
En quoi COM est-il un modèle unificateur ?

- Les problèmes posés par le développement de composants :
  - Interopérabilité
  - Versioning
  - Indépendance vis à vis du langage
  - Transparence interopérabilité cross-process

Comment COM permet-il l 'Interopérabilité ?

Qu 'est-ce qui fait de COM un modèle unificateur véritablement utilisable ?

Pour répondre à ces questions il serait utile de préciser les principes de base du modèle COM et la nature de ses concepts fondamentaux.

Ce faisant, nous étudierons les problèmes spécifiques auxquels COM apporte une réponse.

Il faut commencer par poser les bonnes questions et voir quels sont les problèmes soulevés par le développement de composants :

* interopérabilité. Comment un développeur peut-il s 'assurer que ses composants binaires pourront communiquer avec ceux d 'autres concepteurs ?

* Versioning. Comment maintenir un système de composants sans nécessité la mise à jour de tout le système ?

* Indépendance au langage. Comment faire communiquer des composants écrits dans différents langages ?

* Transparence cross process. Comment créer des composants pouvant tourner In-process ou Cross-process et même Cross-network avec un modèle simple et unique ?

Les bases de COM

- Standard binaire pour les appels entre composants
- Regroupement dans des « Interfaces » de fonctions fortement typées
- Une « Interface » commune de base offrant :
  - le moyen de découvrir les interfaces supportées par un autre composant
  - un compteur de référence permettant aux composants de se détruire eux-mêmes quand ils ne sont plus référencés
  - un procédé permettant d 'identifier un composant de façon unique dans le monde entier
  - un « chargeur de composant » pour mettre en place les interactions entre composants et, additionnellement (cross-process et cross-network) apporter une aide à ces interactions.

Le « modèle COM » définit un ensemble de règles et de mécanismes originaux visant à répondre aux problèmes exposés plus haut.

* Standard binaire pour les appels de fonctions entre les composants.

* la définition d 'Interfaces regroupant des fonctions fortement typées.

* Une Interface commune (minimaliste) offrant :

- l 'accès dynamique aux interfaces supportées par un composant.

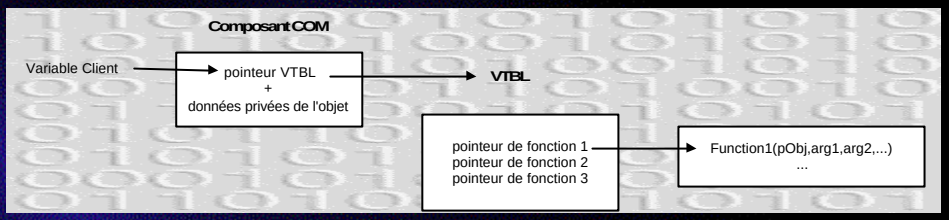
- Compteur de référence pour gérer l 'auto-destruction

- Un système d 'identification garantissant l 'unicité dans le monde.

- Une couche de chargement pour assurer l 'interopérabilité entre les composants et aider cette dernière lors de coopération cross-process ou cross-network.

Le Standard binaire

- COM définit un standard binaire pour chaque plate-forme (couple OS-Hardware)
- Le standard est indépendant du langage

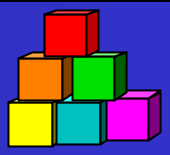


Pour chaque plate-forme (combinaison OS + hardware) COM définit un moyen standard pour accéder aux tables de fonctions virtuelles (vtables) en mémoire et pour faire appel à ces fonctions au travers de la vtable.

Ainsi, tout langage capable d ’appeler ses fonctions par le biais de pointeurs (C, C++, Smalltalk, Ada, Delphi, C++ Builder…) peut être utilisé pour créer des composants COM capable d ’interagir avec d ’autres composants COM écrits dans d ’autres langages.

L ’indirection (le client utilise un pointeur vers la vtable qui est une table de pointeurs vers les fonctions) autorise le partage des vtables en mémoire de plusieurs instances d ’une même classe. Si le système compte des centaines d ’instances d ’objets, ce partage peut réduire l ’utilisation de la mémoire de façon significative.

Objets et Composants



- Component Object Model définit des règles pour des objets.
- Un objet, au sens COM, est un *code compilé offrant des services*.
- L ’accès à un objet passe obligatoirement par une Interface.
- L ’interface primitive est Iunknown

Il semble évident que de par le monde chacun attribue au mot « objet » un sens pouvant différer assez grandement.

Pour clarifier les choses il est nécessaire de préciser que pour COM, est un objet est *un morceau de code compilé qui fournit des services au reste du système.*

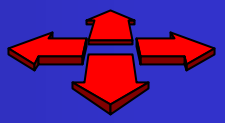
Cette définition évite la confusion avec la définition plus spécifique adoptée en programmation (orientée) objet (C++, Object Pascal notamment).

L ’interface de base commune à tous les objets COM s ’appelle Iunknown. Libre à chaque composant de supporter autant d ’Interfaces (en plus de cette dernière). Cela dépend des fonctions que le composant souhaite exporter, bien entendu.

Il arrive qu ’un objet COM possède des données privées mais à la différences des objets « traditionnels » de la POO jamais un objet COM ne peut avoir accès à un autre objet COM dans sa totalité. En fait cet accès est très limité et fortement typé puisqu ’il passe obligatoirement par des pointeurs vers des Interfaces.

Cette encapsulation plus forte qu ’en POO assure la transparence et offre la possibilité de références distantes transparentes (transparent remoting).

Les Interfaces



- Les applications interagissent entre elles et avec le système au travers de collections de fonctions appelées Interfaces.
- Une Interface est la définition d'un comportement et d'une responsabilité. C'est un contrat.
- Les noms d'Interfaces commence par un « I » (convention comme « T » dans Delphi)

Avec COM, les applications interagissent entre elles et avec le système au travers de collections de fonctions appelées INTERFACES.

On notera que tous les services OLE ne sont simplement que des Interfaces.

Une Interface COM est un « contrat » fortement typé entre les composants conçu pour fournir un ensemble d'opérations (méthodes) sémantiquement liées.

Une Interface est la définition d'un comportement attendu et de responsabilités attendues.

Le support Drag & Drop OLE est un bon exemple. Toutes les fonctions qu'un composant doit mettre en œuvre pour être une cible Drop sont regroupées dans l'Interface IDropTarget. Pour être la source d'un Drag, un composant doit supporter l'Interface IDragSource.

Par convention les Interfaces sont préfixées par « I » ou « IOle ».

Un pointeur vers un objet COM est en réalité un pointeur vers l'une de ses Interfaces. On ne peut donc utiliser un pointeur de composant COM que pour des appels de fonctions et non pour modifier des données.

Spécificités des Interfaces

- Une Interface n'est pas une classe
- Une Interface n'est pas un objet COM
- Les clients COM n'interagissent que par des pointeurs d'Interface
- Un composant COM peut proposer plusieurs Interfaces
- Les Interfaces sont fortement typées.
- Les Interfaces sont immuables.

Une Interface n'est pas une classe. Même si une instance de classe peut être créée (instanciation) pour former un composant COM, une Interface ne peut être instanciée par elle-même parce qu'elle ne possède aucune implémentation. Cette dernière est assurée par le composant COM qui la supporte. Plusieurs composants peuvent implémenter la même Interface de façon différente du moment que le comportement répond au contrat (ex: IStack peut être codé avec une Array ou une liste chaînée).

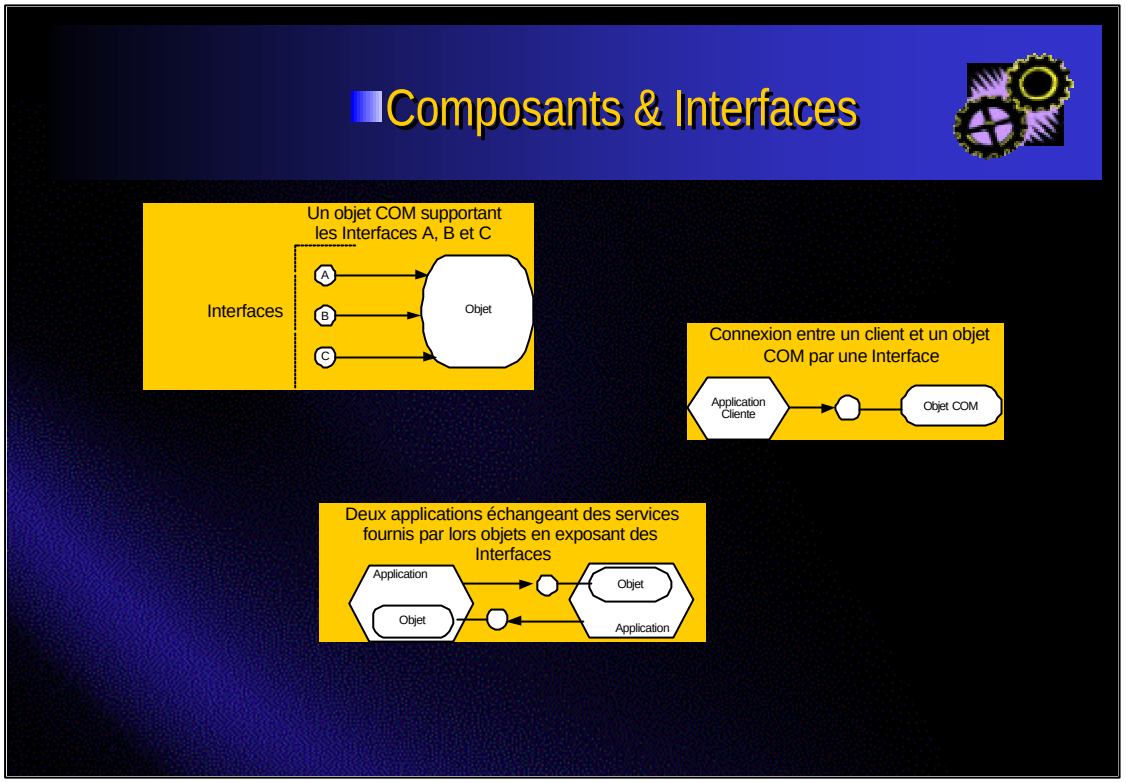
Une Interface n'est pas un composant. Ce n'est qu'une liste de fonctions et un standard au travers du quel les composants interagissent. Un composant COM peut être créé en tout langage et avoir n'importe quelle représentation interne du moment qu'il peut fournir des pointeurs de fonctions.

Les clients COM n'interagissent que par des pointeurs d'Interface. Un client COM ne voit un composant COM que par des pointeurs vers ses Interfaces. Ce sont des pointeurs « opaques » car ils cachent tous les aspects de l'implémentation. C'est cela qui permet le cross-process et le cross-network.

Un composant COM peut proposer plusieurs Interfaces

Les Interfaces sont fortement typées. Chaque Interface à son GUID éliminant les collisions de noms.

Les Interfaces sont immuables. Elles n'évoluent jamais. Toute modification est obligatoire une nouvelle Interface unique (GUID).



L 'utilisation des Interfaces dans COM offre cinq avantages :

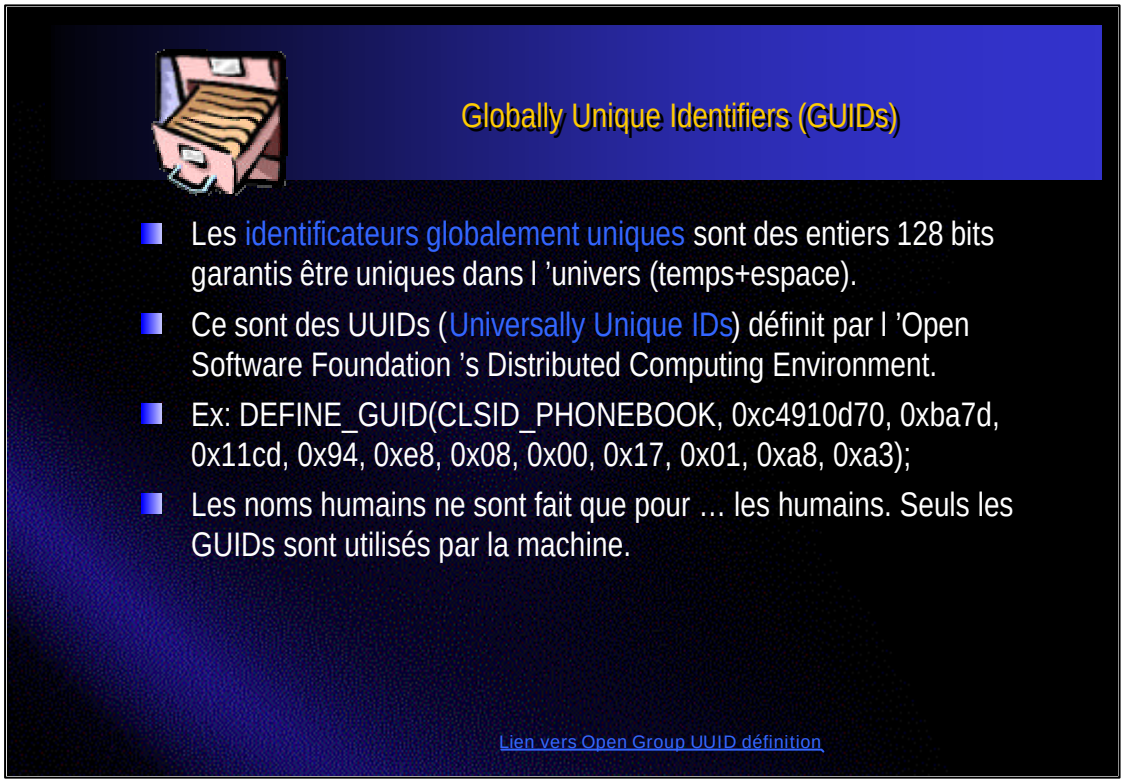
* La possibilité de faire évoluer à tout moment les fonctionnalités d 'une application (client, serveur ou composant COM). Cela se fait par QueryInterface que tous les composants COM supportent et qui permet de proposer de nouvelles fonctions à de nouveaux clients sans briser la compatibilités avec les anciens. Les évolutions se font par de nouvelles Interfaces et un composant COM supporte toujours les vieilles Interface.

* Interactions simples et rapides. Les appels in-process se font par pointeurs (indirection) ce qui ne prend pas plus de temps que les appels aux méthodes virtuelles d 'un objet C++ ou Delphi. Les appels out-process sont impossibles sans COM.

* Réutilisation d 'Interfaces. Les Interfaces, en regroupant des fonctions ayant rapport à un service donné simplifient le support homogène de fonctions identiques.

* Transparence Local/Distant. Le standard binaire autorise le détournement des appels aux Interfaces pour les rediriger vers un objet distant de façon transparente pour l 'appelant.

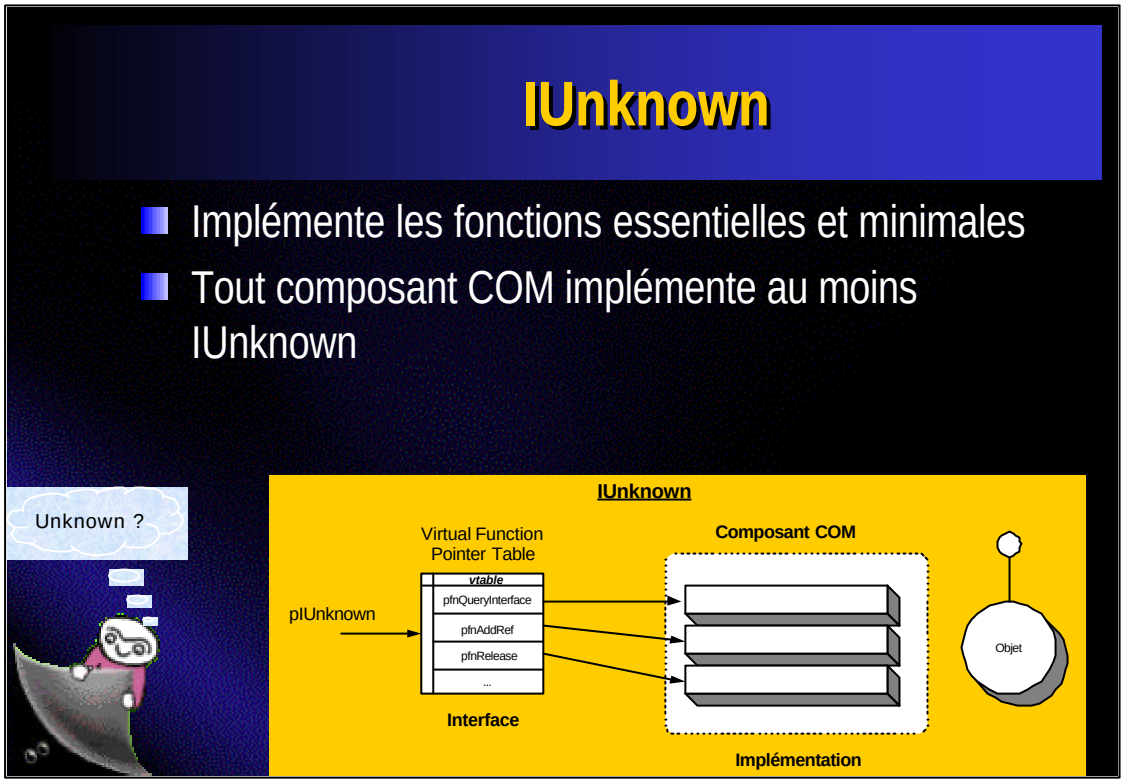
* Indépendance du langage. Tout langage pouvant créer des structures de pointeurs et pouvant appeler des fonctions par pointeur peut utiliser COM et servir à concevoir des composants COM. COM est un standard binaire et contre toute attente il offre une compatibilité que des standards haut-niveau (type langage : échec de la portabilité du C par ex).



Les Interfaces sont identifiées par un code sur 128 bits garanti être unique dans le monde entier.

(128 bits = 16 octets = 8 integer 16 bits ou 4 longint ou 2 Int64)

L 'algorithme de génération d 'un UUID n 'est pas très subtile mais efficace. Pour les curieux j 'ai ajouté un papier de l 'Open Group qui explique la formation d 'un code universel (en lien hypertexte depuis l 'image du slide).



IUnknown est l 'Interface de base que tout component COM doit implémenter, sinon ce n 'est pas un composant COM…

Toutes les autres interfaces COM et OLE sont dérivées de Iunknown.

Cette interface à trois méthodes :

* QueryInterface

* AddRef

* Release

AddRef et Release sont simplement des méthodes qui gère le compteur de référence. AddRef est appelé à chaque fois que le composant COM est utilisé par un client. Release, à chaque fois que le client n 'a plus besoin du composant. Lorsque le compteur de référence passe à zéro, le composant se détruit lui-même.

QueryInterface met en place le mécanisme qui permet à un client de connaître toutes les interfaces supportées par le composant.

Librairie COM

- C 'est un composant système qui met en œuvre les mécanismes de COM
- Elle permet d 'appeler IUnknown en traversant les process.
- Elle encapsule aussi les mécanismes de connexion entre composants

COM se présente sous la forme d 'un composant système qui, une fois installé, autorise le fonctionnement des composants COM sur une machine.

La librairie COM existe pour de nombreuses plate-formes (Windows, NT, Unix, Linux, Macintosh Apple…).

La librairie assure les mécanismes de base de COM (connexion des composants, appels inter-process de IUnknown.

Quand une application crée un composant COM, elle passe le CLSID de ce composant COM à la librairie COM. Cette dernière utilise le CLSID pour chercher le serveur associé dans la base de registre. Si le serveur est un exécutable, COM lance le EXE et attends qu 'il enregistre sa « class factory » par un appel à CoRegisterClassFactory. (Une « classe factory » est un mécanisme qui, dans COM, permet d 'instancier des composants COM).

Si le code est une DLL, COM la charge et appelle DllGetClassFactory.

COM utilise IClassFactory pour créer les instances de composants et renvoyer un pointeur à l 'appelant.

L 'application appelante ne sait jamais où se trouve le serveur.

La librairie COM se trouve dans COMPOBJ.DLL et OLE32.DLL (pour Windows et NT).

Exemple de support COM : les MFC

Microsoft Foundation Class

- L 'équivalent de la VCL pour les développeurs Microsoft
- C 'est une hiérarchie C++ de composants qui supportent COM (comme la VCL)
- C 'est la librairie utilisée de façon naturelle par Windows
- Elle contient de nombreuses classes dont le Rich Edit utilisé dans le WordPad et disponible dans la VCL sous le nom de T(dB)RichEdit.

Les développeurs Microsoft (Visual C++ et MS C++ en général) disposent d 'une librairie de composants équivalente à la VCL de Delphi : les MFC (Microsoft Foundation Class).

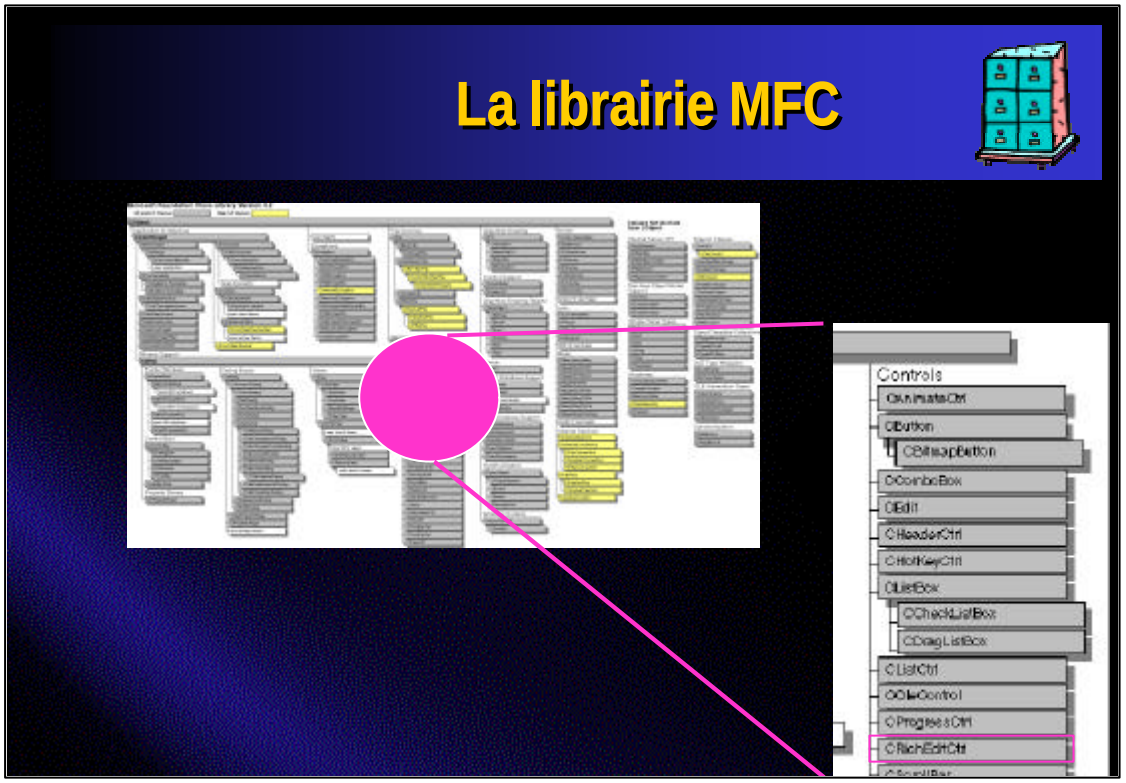
Les relations entre VCL et MFC :

* Les MFC servent de base à la construction d 'applications sous Windows chez Microsoft, application qu 'on peut manipuler par COM depuis Delphi.

* Les classes MFC servent de base aux contrôles Windows comme le Rich Edit, le List box, le Tree View, etc… Autant de classes qu 'on retrouve sous Delphi (TRichEdit, Ttreeview, Tlistbox, …).

Les objets Windows, comme le Rich Edit, supportent des messages spécifiques qui permettent de les manipuler depuis les applications. Mais, en plus, ils supportent COM qui ouvre des portes inaccessibles par le seul système de messages.

Connaître COM et les MFC est donc un atout pour le développeur professionnel qui sait en tirer partie. Nous allons le démontrer avec la création d 'un composant Rich Edit amélioré supportant les objets OLE et les graphismes.



Comme on le voit sur l ’image réduite du slide, la librairie MFC est volumineuse.

Tout comme la VCL elle se divise en plusieurs grandes branches dont les contrôles avec fenêtres et ceux sans fenêtre (dans le sens Windows, avec ou sans Handle).

Tout comme la VCL propose un objet racine (TOBJECT), les classes des MFC dérivent toutes de CObject.

Parmi les contrôles fenêtrés on trouve la classe cRichEditCtrl.

On peut voir les contrôles MFC sous plusieurs angles :

* comme des briques pour construire des programmes d ’une façon proche de celle de Delphi par le biais de la VCL.

* comme des contrôles Windows avec lesquels on dialogue par des messages Windows.

* comme des serveurs COM qu ’on peut contrôler depuis Delphi (ce qui nous intéresse ici).

Le contrôle cRichEditCtrl de la librairie MFC est un composant COM mis en œuvre dans les DLL :

C:\Windows\SYSTEM\RICHED32.DLL ou

C:\Windows\SYSTEM\RICHED.DLL V 4.0

C:\Windows\SYSTEM\RICHED20.DLL V 5.0

Ce contrôle supporte la plupart des Interfaces OLE. Le client est principalement responsable de l 'affichage et du stockage des objets OLE.

Pour obtenir l 'Interface IRichEditOle il faut exécuter un message EM_GETOLEINTERFACE.

Cette interface permet de manipuler le contrôle de façon plus fine et plus efficace que les messages.

Le client peut aussi propose un gestionnaire de stockage en utilisant le message EM_SETOLECALLBACK dans lequel l 'adresse d 'une Interface IRichEditCtrlOleCallBack est passée.

C 'est le support de ce call-back, appelé par le contrôle, qui permet à ce dernier de gérer naturellement les graphiques et les objets OLE...

IRichEditOleCallBack

Interface appelée par le Rich Edit pour déléguer le stockage d 'objets ou informer l 'application.
L 'interface, une fois implémentée, est enregistrée par l 'envoi d 'un message
EM_SETOLECALLBACK.

Méthodes dans l'ordre de Vtable Méthodes de Iunknown	Description
QueryInterface	Retourne un pointeur vers les Interfaces supportées
AddRef	Incrémente le compteur de référence
Release	Décrémente le compteur de référence
méthodes de RichEditOleCallback	**Description**
GetNewStorage	Fournit un stockage pour un nouvel objet collé depuis le presse-papiers ou lu depuis un flux RTF.
GetInPlaceContext	Fournit les interfaces et informations nécessaires au support de l'activation "in-place".
ShowContainerUI	Indique à l'application s'il faut afficher l'interface utilisateur du conteneur.
QueryInsertObject	Demande à l'application si un objet peut être inséré.
DeleteObject	Notifie l'application qu'un objet va être détruit depuis le contrôle Rich Edit.
QueryAcceptData	Appelé sur un "coller" ou un "drag" pour déterminer si les données collée/draggées peuvent être acceptées ou non.
ContextSensitiveHelp	Informe l'application qu'elle doit basculer en ou hors du mode d'aide contextuelle.
GetClipboardData	Permet au client de fournir son propre objet presse-papiers.
GetDragDropEffect	Permet au client de préciser l'effet d'une opération Drop
GetContextMenu	Demande à l'application un menu popup à afficher sur le clic droit de la souris.

Les Call-Back ne sont pas des nouveautés en matière de développement. Ils n 'ont pas forcément porté ce nom dès le départ, mais la technique est vieille.

Sous DOS, les Interruptions du microprocesseur jouaient ce rôle : dans certaines conditions une procédure dont l 'adresse était stockée dans la table des Interruptions était appelée pour prévenir l 'application d 'une action à entreprendre.

Sous Windows cette technique a été généralisée. Par exemple, le BDE propose divers « Call-Back » (Appels arrière). L 'un de ceux-ci informe l 'application lors d 'opérations longues (requêtes ou batchmoves).

Les composants COM utilisent à leur tour cette même technique pour obtenir ou fournir des informations de/à l 'application lorsque cela est nécessaire.

Comme les composants COM repose sur les principes rigides de COM exposés plus avant, les call-backs ne pointent pas directement vers l 'application mais vers une Interface qui, du côté Delphi, est matérialisée en écrivant une classe dérivée de TInterfacedObject supportant l 'interface call-back requise.

Implémentation Delphi du Call-Back

Delphi 3, et plus encore 4, supporte fortement le modèle COM et propose des classes spécialisées simplifiant l 'écriture du code.

```
TRichEditOleCallback = class(TInterfacedObject, IRichEditOleCallback)
  private
    FOwner: TCustomRichEdit98;
  protected
    function GetNewStorage(out stg: IStorage): HRESULT; stdcall;
    function GetInPlaceContext(out Frame: IOleInPlaceFrame;
         out Doc: IOleInPlaceUIWindow; var FrameInfo: TOleInPlaceFrameInfo): HRESULT; stdcall;
    function ShowContainerUI(fShow: BOOL): HRESULT; stdcall;
    function QueryInsertObject(const clsid: TCLSID; stg: IStorage; cp: longint): HRESULT; stdcall;
    function DeleteObject(oleobj: IOLEObject): HRESULT; stdcall;
    function QueryAcceptData(dataobj: IDataObject; var cfFormat: TClipFormat;
         reco: DWORD; fReally: BOOL; hMetaPict: HGLOBAL): HRESULT; stdcall;
    function ContextSensitiveHelp(fEnterMode: BOOL): HRESULT; stdcall;
    function GetClipboardData(const chrg: TCharRange; reco: DWORD;
         out dataobj: IDataObject): HRESULT; stdcall;
    function GetDragDropEffect(fDrag: BOOL; grfKeyState: DWORD;
         var dwEffect: DWORD): HRESULT; stdcall;
    function GetContextMenu(seltype: Word; oleobj: IOleObject;
         const chrg: TCharRange; var menu: HMENU): HRESULT; stdcall;
  public
    constructor Create(AOwner: TCustomRichEdit98);
  end;
```

La VCL définit une classe simple, TInterfacedObject qui, pratiquement, sert de classe de base car elle implémente les méthodes de IUnknown. La classe TInterfacedObject est déclarée de la manière suivante dans l'unité System :

```
type
  TInterfacedObject = class(TObject, IUnknown)
  private
    FRefCount: Integer;
  protected
    function QueryInterface(const IID: TGUID; out Obj): Integer; stdcall;
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;
  public
    property RefCount: Integer read FRefCount;
  end;
```

On choisit cette classe comme ancêtre pour mettre en œuvre le code de l 'Interface du call-back OLE du contrôle Rich Edit.

GetNewStorage

GetNewStorage doit être codé pour autoriser les opérations Couper/Copier/Coller et Drag 'n Drop d 'objets OLE.

```
function TRichEditOleCallback.GetNewStorage(out stg: IStorage): HRESULT;
var LockBytes: ILockBytes;
begin
  Result:= S_OK;
  try
    OleCheck(CreateILockBytesOnHGlobal(0, True, LockBytes));
    OleCheck(StgCreateDocfileOnILockBytes(LockBytes, STGM_READWRITE
      or STGM_SHARE_EXCLUSIVE or STGM_CREATE, 0, stg));
  except
    Result:= E_OUTOFMEMORY;
  end;
end;
```

Cette fonction de l 'Interface IRichEditOleCallback est la clé de voûte ouvrant les perspective du stockage d 'objets OLE et de graphiques ou autres en provenance du presse-papiers.

Il est vrai que seule cette fonction mérite d 'être codée pour obtenir le but que nous nous sommes fixé (Rich Edit avec graphiques et objets OLE).

Il est un peu déconcertant de s 'apercevoir que ces quelques lignes n 'ont pas été codées dans la VCL.

Dans le même temps on perçoit aussi la grande complexité des concepts mis en jeu pour coder cette petite fonction (2 appels à des procédures OLE).

Ceci expliquant peut-être cela...

OleCheck

OleCheck déclenche une exception EOleSysError si le code de résultat indique une erreur.

Unité

ComObj

Catégorie

utilitaires COM

```
procedure OleCheck(ECode: HResult);
```

OleCheck est une procédure qui contrôle le code retourné par les fonctions OLE.

En cas d ’erreur elle déclenche une exception DELPHI qui peut ainsi être traitée de façon cohérente avec le modèle proposé par Delphi (Try/Except/Finally).

CreateILockBytesOnHGlobal

```
{$EXTERNALSYM CreateILockBytesOnHGlobal}
function CreateILockBytesOnHGlobal(hglob: HGlobal;
fDeleteOnRelease: BOOL;
  var lkbyt: ILockBytes): HResult; stdcall;
{ Externals from ole32.dll }
```

ILockBytes est une interface utilisée par les documents composites OLE pour obtenir une zone de stockage physique pour les objets imbriqués.

La fonction CreateILockBytesOnHGlobal crée une instance de l 'Interface ILockBytes en mémoire.

L 'Interface IlockBytes fournit un objet définissant un tableau d 'octet (byte array) utilisé dans l 'implémentation des fichiers composites OLE. Ses méthodes permettent de manipuler le tableau qui fournit le stockage physique des objets contenus dans un fichier composite OLE. Le tableau d 'octets de ILockBytes peut être implémenté de diverses manières : fichier disque, mémoire globale une même base de données.

L 'Interface, selon le concept COM permet de masquer totalement les détails des accès au support physique.

La fonction CreateILockBytesOnHGlobalcrée une interface ILockBytes.

```
HRESULT CreateILockBytesOnHGlobal(
  HGLOBAL hGlobal,          //Memory handle for the byte array object
  BOOL fDeleteOnRelease,    //Indicates whether to free memory when the object is released
  ILockBytes ** ppLkbyt     //Points to location for pointer to the new byte array object
 );
```

Paramètres :

hGlobal

Contient le Handle de mémoire alloué par la fonction GlobalAlloc. Le Handle doit être créé en mode Moveable et NonDiscartdable. Si le Handle est supposé être partagé par plusieurs process, il doit être alloué en mode partagé (shared). Les nouveaux Handles doivent être alloué avec une taille à zéro. Si hGlobal est nul, la fonction alloue en interne un nouveau block mémoire de taille nulle.

fDeleteOnRelease

Indique si le Handle sous-jacent pour le tableau d 'octets doit être automatiquement libéré quand l 'objet lui-même est supprimé.

ppLkbyt

Pointe vers l 'emplacement où la fonction doit ranger le pointeur du tableau d 'octets créé, c 'est à dire, la nouvelle instance de l 'Interface ILockBytes.

Valeurs retournées

S_OK Indique que l 'interface ILockBytes a été créée correctement

E_INVALIDARG Une valeur incorrecte a été passée au paramètre hGlobal

E_OUTOFMEMORY Il n 'y a pas assez de mémoire pour créer le tableau

StgCreateDocfileOnILockBytes

StgCreateDocfileOnILockBytes crée et ouvre un nouvel objet de stockage au sommet d 'une instance ILockBytes fournie par l 'appelant.

```
HRESULT StgCreateDocfileOnILockBytes(
    ILockBytes * plkbyt,      //Specifies the byte array object
    DWORD grfMode,            //Specifies the access mode
    DWORD reserved,           //Reserved; must be zero
    IStorage ** ppstgOpen     //Points to location for returning the new storage object
   );
```

StgCreateDocfileOnILockBytes crée et ouvre un nouvel objet de stockage au sommet d 'une instance ILockBytes fournie par l 'appelant.

```
HRESULT StgCreateDocfileOnILockBytes(
  ILockBytes * plkbyt,      //Specifies the byte array object
  DWORD grfMode,            //Specifies the access mode
  DWORD reserved,           //Reserved; must be zero
  IStorage ** ppstgOpen     //Points to location for returning the new storage object
 );
```

Paramètres

plkbyt

Pointe vers l 'Interface ILockBytes sur laquelle sera créé le fichier composite.

grfMode

Spécifie le mode d 'accès à utiliser lors de l 'ouverture du nouveau fichier composite. (voir le type énuméré STGM).

reserved

Réservé pour le futur (doit être à zéro)

ppstgOpen

Pointe vers l 'emplacement où le nouvel objet de stockage est placé.

Valeurs retournées

S_OK : indique le succès (fichier composite créé)
STG_E_ACCESSDENIED : L 'appelant n 'a pas les droits suffisants (ou conflit sur même objet ouvert par un autre process).
STG_E_FILEALREADYEXISTS : Le fichier existe déjà et grfMode est passé à STGM_FAILIFTHERE.
STG_S_CONVERTED : Le fichier a été correctement converti (le tab leau d 'octets a été converti au format Istorage).
STG_E_INSUFFICIENTMEMORY : pas assez de mémoire
STG_E_INVALIDPOINTER : pLkbyt ou ppStgOpen est un pointeur non valide
STG_E_INVALIDFLAG : la combinaison deflags dans grfMode est incorrecte
STG_E_TOOMANYOPENFILES : trop de fichiers ouverts

Commentaires

Cette fonction crée un objet de stockage sur le sommet d 'un tableau d 'octets en utilisant l 'implémentation fournie par OLE de l 'Interface Istorage. La fonction peut être utilisée pour stocker un document dans un SGBD-R. Le tableau d 'octets passé en paramètre est utilisé pour stocker le fichier en place et lieu du disque.

Le nouveau fichier créé est ouvert en fonction du mode d 'accès dans grfMode.

Les autres méthodes de TRichEditOleCallBack

Il n 'est pas nécessaire de coder toutes les méthodes d 'une interface (cela dépend du contexte tout de même).
La constante E_NOTIMPL permet de dire que la fonction n 'est pas supportée.
Si aucun traitement ne se justifie, la fonction peut aussi retourner directement S_OK.

```
function TRichEditOleCallback.ShowContainerUI(fShow:
BOOL): HRESULT;
begin
  Result:= E_NOTIMPL;
end;

function TRichEditOleCallback.QueryInsertObject(const
clsid: TCLSID; stg: IStorage;
       cp: longint): HRESULT;
begin
  Result:= S_OK;
end;
```

Non implémenté

Aucun traitement

Les autres méthodes de l 'objet supportant l 'interface IRichEditOleCallBack ne nécessitent pas toutes d 'être codées.

Par exemple, ShowContainerUI retourne la constante E_NOTIMPL signifiant que cette fonction n'est pas implémentée.

De la même façon, la méthode xxx ne code aucun traitement particulier. Ici nous acceptons toujours les objets que l'utilisateurs intègre à son document. De fait, le code se limite à retourner la constante S_OK.

Bien que la mécanique COM / OLE ne soit pas évidente, il s'avère que dans la pratique elle n'est pas non plus si compliquée.

On peut même dire que les principes mis en œuvre dans COM sont assez simples puisqu'il s'agit de conventions, de règles qu'il suffit d'appliquer. Le véritable travail est un investissement long de recherche afin de maîtriser les nombreuses Interfaces et leurs méthodes.

IRichEditOle

Interface obtenue par le message EM_GETOLEINTERFACE

IRichEditOle
Spécifie une Interface utilisée par le client d'un contrôle Rich Text Edit pour supporter les opérations OLE. Cette Interface à les méthodes suivantes:

Méthodes dans l'ordre de la Vtable

Méthodes de IUnknown	Description
QueryInterface	Retourne un pointeur vers les interfaces supportées
AddRef	Incrémente le compteur de référence
Release	Décrémente le compteur de référence

Méthode de IrichEditOle	Description
GetClientSite	Récupère une Interface IoleClientSite à utiliser lors de la création d'un nouvel objet.
GetObjectCount	Retourne le nombre d'objets contenus dans le contrôle RTF
GetLinkCount	Retournes le nombre d'objets contenus dans le contrôle RTF qui sont des liens.
GetObject	Retourne les informations sur un objet dans une structure REOBJECT
InsertObject	Insèrfe un objet dans le contrôle RTF
ConvertObject	Convertit un objet dans un nouveau type
ActivateAs	Décharge les objets de l'ancienne classe et dit à OLE de traiter ces objets comme étant de la nouvelle classe, puis recharge tous les objets.
SetHostNames	Renseigne les "host names" qui sont donnés aux objets quand ils sont insérés dans le contrôle RTF.
SetLinkAvailable	Renseigne le bit "link disponible" dans les flags de l'objet.
SetDvaspect	Indique dans quel mode un objet doit être représenté dans le contrôle.
HandsOffStorage	Demande au contrôle de libérer ses références à l'interface de stockage associée à l'objet spécifié.
SaveCompleted	Indique au contrôle RTF que l'opération de sauvegarde la plus récente est terminée et qu'il uriliser le nouveau stockage passé en paramètre.
InPlaceDeactivate	Demande au contrôle de désactiver l'objet en cours d'édition in-place s'il y en a un.
ContextSensitiveHelp	Indique au contrôle s'il doit basculer ou non en mode Aide contextuelle
GetClipboardData	Récupère un objet presse-papiers pour une Range dans le contrôle
ImportDataObject	Importe l'objet presse-papiers dans le contrôle en remplaçant la sélection active.

Voici l ’interface IRichEditOle.

On obtient un pointeur vers elle en envoyant au contrôle RichEdit le message EM_GETOLEINTERFACE.

Une fois l ’interface obtenue, on accède à l ’ensemble de ces méthodes.

Le contrôle du stockage des objets, l ’activation du mode d ’aide contextuelle, l ’accès à tous les objets imbriqués ou liés, tout peut être commandé par IRichEditOle.

Exemple d 'utilisation de l 'Interface IRichEditOle

Un exemple simple d 'exploitation de IRichEditOle :
Savoir si un objet imbriqué ou lié est actuellement sélectionné dans le contrôle par l 'utilisateur.

On passe par l 'Interface pour obtenir l 'objet en cours de sélection puis on regarde si la référence retournée est valide.

TReObject est une structure utilisée pour retourner des informations sur les objets.

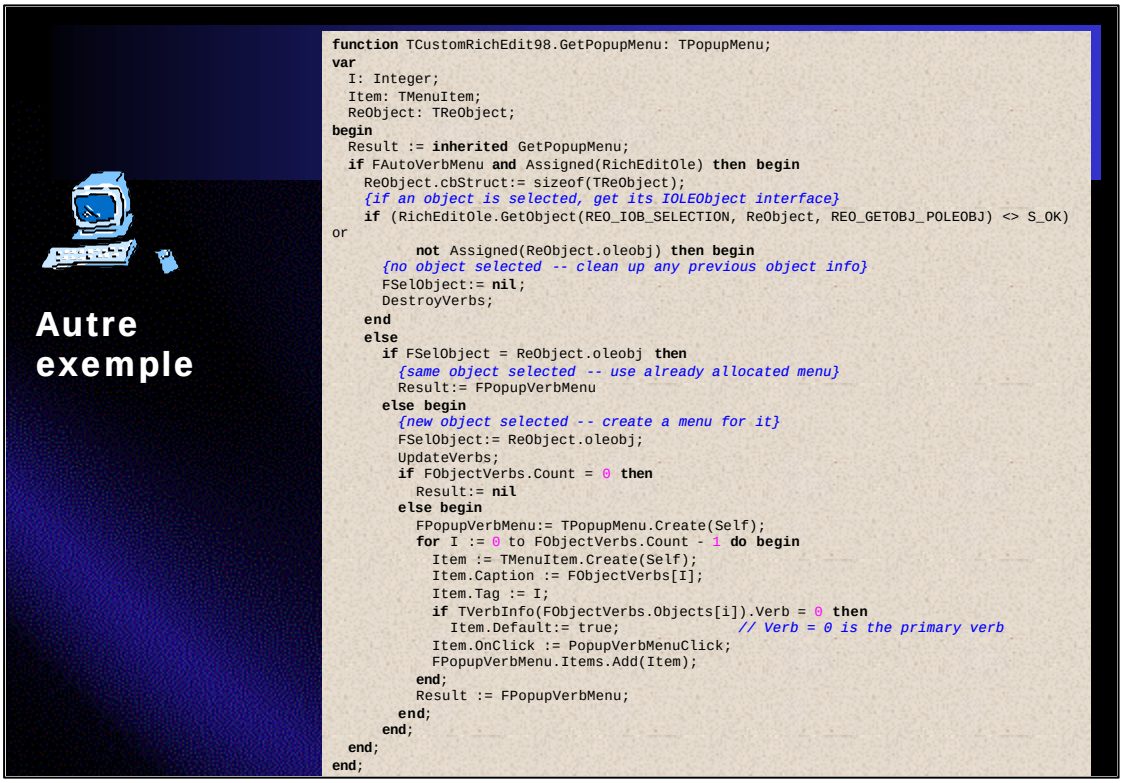
```
function TCustomRichEdit98.ObjectSelected:Boolean;
var ReObject:TReObject;
begin
  ReObject.cbStruct:= sizeof(TReObject);
  result:=(RichEditOle.GetObject(REO_IOB_SELECTION, ReObject,
REO_GETOBJ_POLEOBJ) = S_OK) and
          Assigned(ReObject.oleobj);
end;
```

Voici un exemple assez simple de l 'utilisation de IRichEditOle.

L a classe TRichEdit98 expose une propriété permettant au développeur de savoir à tout moment si un objet lié ou imbriqué est actuellement sélectionné par l 'utilisateur.

La technique mise en œuvre est triviale : par le biais de l 'Interface on demande à obtenir les informations de l 'objet en cours de sélection. Ces informations sont passées dans une structure (Record en Pascal) que nous initialisons avant l 'appel (en indiquant sa taille. Cette technique est fréquente sous Windows, elle permet de faire évoluer la fonction qui peut retourner plus d 'information. En lisant la taille de la structure passée en paramètre la fonction sait ce qu 'elle doit renvoyer).

Nous contrôlons ensuite si la référence vers l 'Interface de l 'objet est valide ou non.



Cet exemple est plus complexe. En effet, nous allons utiliser l ’interface IRichEditOle (ainsi que lIRichEditOleCallBack) pour gérer un popup local pour chaque objet lié ou imbriqué.

Plusieurs fonctions et procédures sont utilisées pour mettre en œuvre la totalité du mécanisme mais tout se justifie de façon assez simple.

Ici, nous représentons le code de la fonction GetPopupMenu qui retourne un menu pour l ’objet en cours.

Le code est commenté et ne pose pas de problème (des références sont faites à des champs et des méthodes de la classe non représentés ici mais les noms suffisent à en imaginer le rôle).

Pour plus de détail, reportez-vous au code source complet...

C 'est fini ...

Mais vous pourrez retrouver d'autres documents en allant sur le site de l'auteur :

Email : odahan@cybercable.fr
Web : http://perso.cybercable.fr/stargate/

Et en visitant : www.developpez.com