UTILISATION DE DSPACK 234 SOUS DELPHI 2005

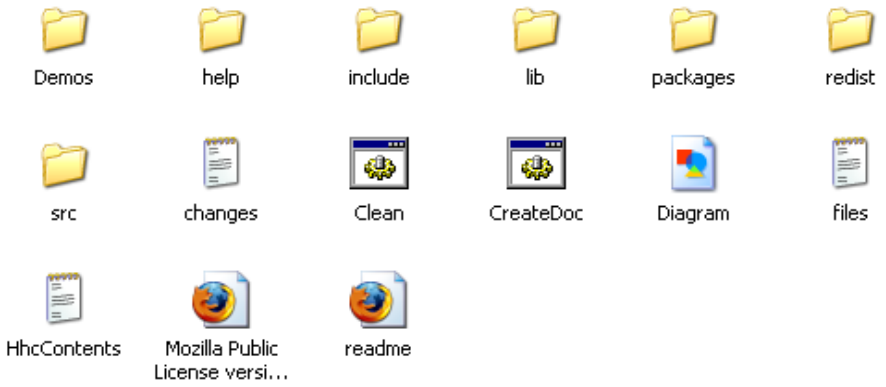
TABLE DES MATIERES

I. INTRODUCTION

A travers ce tutoriel je vais vous montrer comment utiliser le pack de composants DSPACK afin de créer des applications ***DirectShow*** sous Delphi. Ce projet a été créé par la communauté ***SourceForge*** et il est totalement gratuit. Il est composé d'un certain nombre de composants permettant de faciliter l'accès à ***DirectShow***. Nous en verrons les principaux durant l'étude de ce tutoriel.
Afin de bien appréhender la programmation ***DirectShow*** sous Delphi, j'ai décomposé ce tutoriel en cinq principales grandes parties. Tout d'abord nous verrons quels sont les pré requis pour pouvoir utiliser le DSPACK sous Delphi, c'est-à-dire les éléments essentiels à télécharger et à installer sur votre disque dur.
Nous verrons ensuite qu'il est nécessaire d'apporter quelques modifications aux sources d'origine pour pouvoir installer le pack sous Delphi 2005 dans de bonnes conditions. Nous verrons ensuite comment il est possible de concevoir un graphe à la main grâce au logiciel ***GraphEdit*** sans taper une ligne de code. Je vous donnerai toutes les informations pour créer votre premier programme. Nous verrons ensuite des notions plus avancées vous permettant de diffuser un fichier via un périphérique externe, en créant un graphe de façon dynamique. Nous finirons par une conclusion. Normalement à la fin de ce tutoriel vous devriez avoir les notions essentielles afin de créer votre propre Media Player.

II. PREREQUIS

Afin de pouvoir utiliser dans de bonnes conditions le pack en question il va falloir installer un certain nombre de SDK. A commencer bien entendu par le pack lui-même que vous pourrez télécharger à partir du site www.progdigy.com en cliquant sur le lien DSPACK. Après l'avoir téléchargé, installez le sur votre PC. Moi je l'ai installé dans le répertoire …\Borland\BDS 3.0\ pour information. Voici ce que vous devriez obtenir :

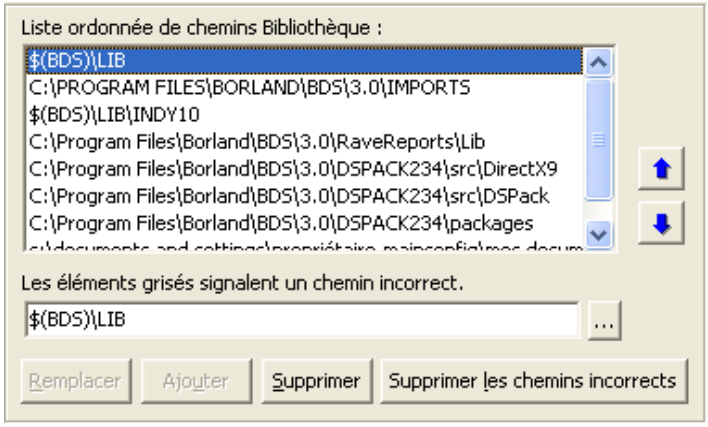


Il vous faut ensuite installer le DirectX SDK de Microsoft version 9.0 (Summer 2004). Une fois installé sur votre disque dur il va falloir donner à Delphi l'emplacement des bibliothèques de fonctions DSPACK, comme suit :

- Cliquez sur Outils->Options
- Faites dérouler la liste Options d'environnement -> Options Delphi -> Bibliothèque Win32

- Cliquez sur Ajouter…
- Ajouter les chemins d'accès de DSPACK.

Vous devez obtenir ceci :



Une fois cette phase terminée vous devez apporter quelques modifications au projet DSPACK. En effet ce pack a été créé à l'origine pour être utilisé à partir des versions 5, 6 et 7 de Delphi. Donc il n'est pas possible d'installer directement le pack et l'utiliser tel quel. Pour vous en convaincre ouvrez le projet DirectX9_D7.bds. Il vous demande sous quelle plateforme vous voulez convertir le projet. Convertissez le projet sous la plateforme Win32, Compilez le projet. Une série d'erreur se produit.
Il faut donc apporter un certain nombre de changement au code de DirectX9_D7.

Un certain nombre de modifications sont à effectuer pour pouvoir utiliser le pack...

(1)Dans le fichier DirectDraw.pas replacer :

PDirectDrawSurface = IDirectDrawSurface

par :

PDirectDrawSurface = ^IDirectDrawSurface

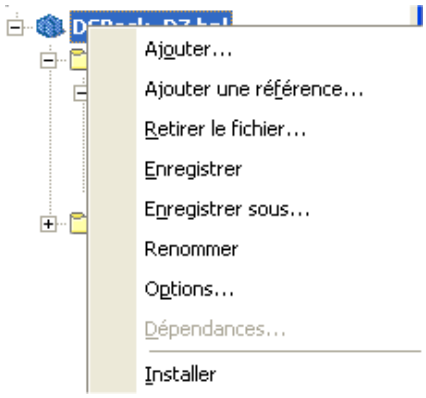
(2)Dans ***DSUtil.pas*** Ajouter l'unité variants dans la clause uses

(3) A la ligne 3072 remplacer ***result := nil***; par ***result := NULL***;

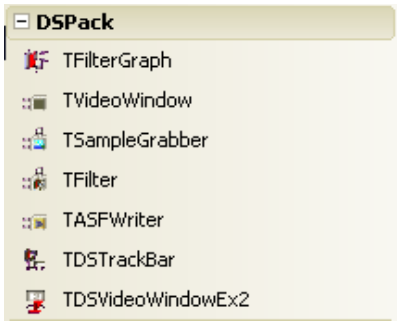
(4) Enfin il faut rajouter dans les uses de ***DSeditors*** cette ligne :

{$IFDEF VER170} DesignIntf, DesignEditors, {$ELSE}

Installez ensuite le package proprement dit dans la liste des composants, pour ce faire il suffit de cliquer droit sur le projet .bpl et choisir la commande Installer, comme ceci :



Si tout s'est bien déroulé vous devriez voir apparaître dans la liste des composants toute une série dans la catégorie DSPACK. Comme ceci :



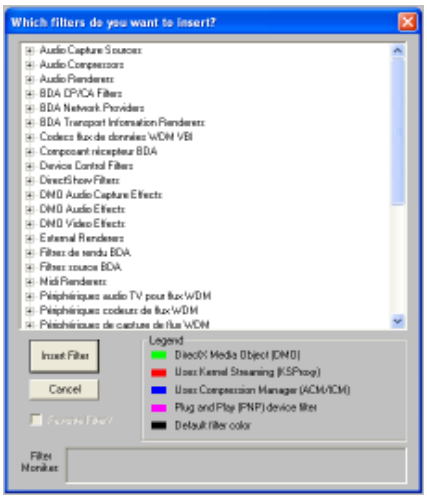
Voilà maintenant vous êtes prêt à développer votre première application multimédia.
Cependant avant de commencer le développement d'un telle application vous allez avoir besoin d'un certain nombre de filtres et de codecs, pour pouvoir gérer le plus grand nombre de formats possibles.
Ceux-ci peuvent être installé grâce à l'installation d'applications comme des Media Player (Windows Media Player, VLC, ou encore Media Player Classic que je vous recommande), ou d'applications de lecture de DVD (WInDVD par exemple), mais aussi de montage, ou encore après l'installation d'un périphérique d'acquisition ou de diffusion comme la Pinnacle PCTV/USB me concernant…
Si vous voulez des codecs gratuits je vous en propose deux qui sont téléchargeables sur ZDNET :

- K-Lite Codec
  http://www.zdnet.fr/telecharger/windows/fiche/0,39021313,39056680s,00.htm
- Matroska Pack Full
  http://www.zdnet.fr/telecharger/windows/fiche/0,39021313,39067760s,00.htm

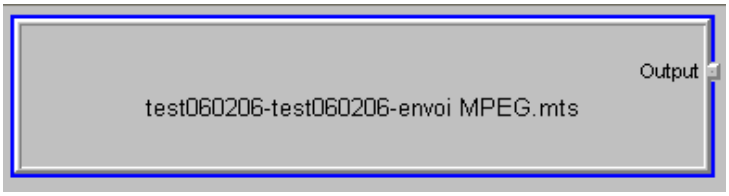
Normalement avec ceci vous devriez commencer votre première application sur de bonnes bases.

III. GRAPHEDIT PREMIERE APPROCHE DE DIRECTSHOW

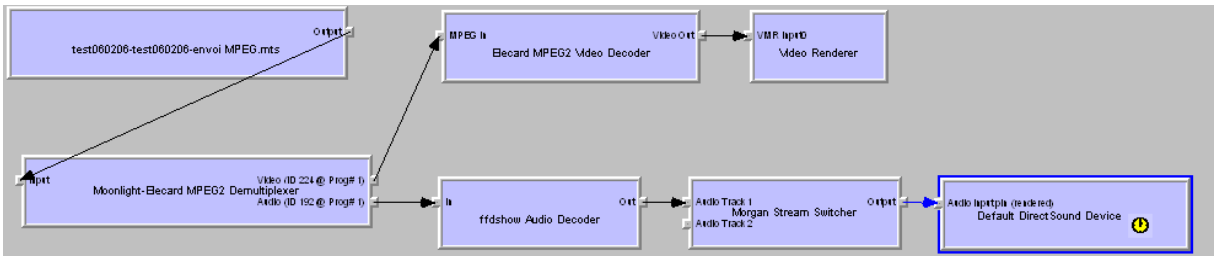
Nous allons utilisé pour cette première approche, le logiciel ***GraphEdit***, fourni avec le DirectX SDK. Ce logiciel permet de créer une représentation sous forme de graphes permettant la lecture d'un fichier ***mpg*** par exemple.
Lancez l'application (Démarrer->Programmes->DirectX SDK->Utilities->GraphEdit).
Nous allons créer un graphe nous permettant de visualiser de visualiser un fichier MPEG.
Pour ce faire aller dans le menu ***Graph***, cliquez sur la commande ***Insert Filters***. Là vous devriez voir apparaître la fenêtre suivante à l'écran :



Cette fenêtre contient tous les filtres installés sur votre disque dur, ils sont répertoriés par catégorie. Pour insérer un filtre à l'écran il suffit de double-cliquer sur celui-ci ou d'appuyer sur le bouton OK. Le filtre qui nous intéresse pour ouvrir un fichier est le filtre File Source (Async.) qui se trouve dans la catégorie ***DirectShow Filters***. Une fenêtre vous demande choisir le fichier que vous voulez visualiser. Une fois inséré sur la zone cliente, vous devriez voir apparaître cette figure à l'écran :



Comme vous pouvez le constater un filtre peut être assimilé à un « composant électronique » qui effectue une fonction bien précise, dans le cas présent il nous permet d'ouvrir un fichier multimédia et d'envoyé un flux de données via la broche de sortie. Donc comme on peut le constater un filtre est formé de même de broche (ou pin en anglais), qui vont l'aider à accomplir sa fonction. Vu comme tel le graphe produit ne va pas réaliser grand chose de concluant, il lui manque une série de filtre pour pouvoir arriver à notre fin, c'est-à-dire la visualisation du media. Pour finaliser le graphe il suffit de relier la sortie de notre filtre à une série de filtres, qui par leur fonction particulière vont permettre d'aboutir au résultat escompté. Pour ce faire il suffit de cliquer droit sur la broche Output de notre File Source, et dans le ***popup*** cliquer sur la commande ***Render Pin***. Et là par miracle, le graphe se complète en fonction des filtres installés sur votre ordinateur.
Sur mon ordinateur le résultat obtenu est le suivant, mais il peut varier en fonction des éléments installés.



Donc on peut dire que ***DirectShow*** sait gérer intelligemment la liaison entre les filtres, et comme je le disais avant quelque soit les éléments installé. Ce procédé va nous permettre ensuite bien sûr une programmation plus aisée de nos applications. Pour vous en convaincre, et afin de vérifier si toutes les connexions sont bien effectuées on va mettre en lecture le graphe. Il suffit de cliquer le bouton Play pour voir la lecture du fichier via une fenêtre Active ***Movie***. Vous pouvez recommencer l'opération en choisissant un autre type de fichier video (par exemple un avi). Et vous vous apercevrez que le graphe change d'aspect, c'est-à-dire que les filtres intermédiaires sont différents.

Pour conclure il faut garder à l'esprit que ***DirectShow*** fonctionne à partir d'un graphe composé de filtres, accomplissant des tâches particulières, reliés entre eux par des broches comme des « composants électroniques » en schématisant bien sûr.

Maintenant que nous avons vu comment ***DirectShow*** gérer la lecture d'un fichier multimédia. Nous allons voir comment nous pouvons le coder sous Delphi.

IV. PREMIER PROGRAMME UTILISANT DSPACK

Afin de bien commencer dans ce domaine, j'ai eu la chance d'avoir à ma disposition un certain nombre d'informations très intéressantes à travers le Web. Je vais vous donner quelques sites à explorer sans aucune modération. Tout d'abord je vous conseille le site de l'auteur de ce pack http://www.progdigy.com. Celui-ci vous apportera une mine d'information, et il donne accès à un forum, qui vous permettra de poser vos questions concernant l'utilisation de ce pack, ainsi que sur ***DirectShow*** en général. L'inconvénient de ce site est qu'il est totalement en anglais.

Le second site, en français, permet d'appréhender dans de bonnes conditions le monde du multimédia, sous Delphi, et dont voici l'URL : http://paul.glagla.free.fr.
Ce site vous permettra de vous familiariser avec la conversion, la prévisualisation ou l'acquisition de média audiovisuels. Moi ce que je vous propose de réaliser, c'est la diffusion d'un média à travers un périphérique de sortie vidéo.

V. NOTIONS AVANCEES – REALISATION D'UNE FONCTION DE DIFFUSION

Nous allons voir dans cette partie quelques notions un peu plus avancées concernant la programmation ***DirectShow***.
Nous allons le voir par l'intermédiaire d'un exemple. Nous allons concevoir un graphe nous permettant de diffuser le fichier via une destination externe. C'est-à-dire qu'au lieu d'avoir une ***videoWindow*** comme maillon de fin de chaîne nous aurons un autre ***VideoRenderer***. Pour ma part, j'ai utilisé ma ***Pinnacle PC/TV USB***. Cette rubrique va nous permettre de créer un graphe. Celle-ci va donc nous permettre de voir comment ***DirectShow*** identifie des filtres, et nous verrons ensuite comment ajouter un filtre à un graphe. Puis nous verrons ensuite comment interconnecter chaque filtre entre eux.

a. Préparatifs

Nous allons placer sur notre fiche principale un objet ***TFilterGraph*** qui va nous permettre de géerer la conception de notre graphe de diffusion.
Une fois que vous avez placé votre objet, vous pouvez le nommer ***fgRenderer*** par exemple.
Le reste des objets va être déclaré dynamiquement au cours de la conception.

b. Déclarations des interfaces nécessaires

Tout d'abord il faut interfacer notre ***filterGraph*** comme étant dérivé de l'interface ***IGraphBuilder*** (***fgRender as IGraphBuilder***), cela va nous permettre de construire notre graphe dynamiquement, et en particulier de créer une instance des interfaces ***IMediaControl*** et ***IMediaEvent*** qui vont nous permettre d'interagir sur le media. Voilà les lignes de code nécessaire :

(fgRender as IGraphBuilder).QueryInterface(IID_IMediaControl, mc);

Et

(fgRender as IGraphBuilder).QueryInterface(IID_IMediaEvent, mEvent);

Bien entendu il faut déclarer les variables ***mEvent*** et ***mc*** comme ceci :

mEvent : IMediaEvent;
mc : IMediaControl;

Maintenant que nous avons déclaré ces instances d'interfaces il nous faut construire le graphe qui va nous permettre de diffuser notre media.

c. Ajout dynamique d'un filtre dans un graphe

Tout d'abord il va nous falloir insérer dans notre graphe un filtre source qui va nous permettre de charger le fichier multimédia. Dans le jargon ***DirectShow*** ce filtre s'appelle « Source filter » qui correspond en fait au filtre Source Filter (Async.) que l'on a utilisé dans GraphEdit au début de ce tutoriel.
Pour l'insérer dans le graphe il suffit d'appeler la fonction AddSourceFilter comme ceci :

hr := (fgRender as IGraphBuilder).AddSourceFilter(StringToOleStr(fichier), 'Source Filter', SourceFilter);

Cette fonction prend en paramètre :

- Le chemin du fichier à ouvrir ;
- Le nom du filtre tel qu'il va apparaître dans ***GraphEdit*** ;
- ***SourceFilter*** étant l'objet ***IBaseFilter*** ;

En fait lorsque l'on construit le graphe on insère des objets ***IBaseFilter***. Vous pouvez voir de même que dans les propriétés d'un objet ***Tfilter***, il y a une propriété ***BaseFilter***.

Nous allons maintenant insérer le filtre de sortie, c'est-à-dire le filtre de diffusion. Celui-ci diffère du cas précédent car on visualisait le media sur la fenêtre de notre application via un ***TVideoWindow***. Maintenant dans notre cas présent nous voulons que notre media soit diffusé via une source externe, c'est-à-dire par l'intermédiaire de la ***Pinnacle PCTV/USB***. Cette opération est beaucoup moins simple que d'ajouter un filtre source. En effet il va falloir aller fouiller dans toute la liste des filtres. Pour ce faire, il faut utiliser le code suivant :

```
SysDev:= TSysDevEnum.Create;

 for i:= 0 to SysDev.CountCategories-1  do
   begin
    if SysDev.Categories[i].FriendlyName = sVideoRenderCategory then
      begin
        G:= SysDev.Categories[i].CLSID;
        FilterEnum:= TSysDevEnum.Create(G);
        for j:= 0 to FilterEnum.CountFilters-1 do
          begin
            if FilterEnum.Filters[j].FriendlyName = sVideoRender then
              begin
                CLSID_VideoRender:= FilterEnum.Filters[j].CLSID;
                break;
              end;

          end;
      end;

   end;
SysDev.Free;
FilterEnum.Free;


 hr := CoCreateInstance(CLSID_VideoRender, nil, CLSCTX_INPROC_SERVER,
      IID_IBaseFilter, IVideoRenderFilter);
(fgRender as IGraphBuilder).AddFilter(IVideoRenderFilter, 'Pinnacle Video Render');
```

Donc comme on peut le constater on va créer une instance de la classe SysDevEnum qui va nous permettre de scruter notre liste de filtres. Tout d'abord disons que notre liste est rangée

par catégorie et donc il va falloir scruter notre liste en fonction de ce paramètre. Nous le faisons par l'intermédiaire de la boucle suivante

```
for i:= 0 to SysDev.CountCategories-1  do
   begin
   ….
  End
```

On teste ensuite les différentes catégories avec celle qui nous intéresse :

```
if SysDev.Categories[i].FriendlyName = sVideoRenderCategory then
```

Dès que le test est bon on va chercher dans la liste de cette catégorie le filtre qui nous intéresse. Pour ce faire on scrute la sous liste en fonction de son ID :

```
G:= SysDev.Categories[i].CLSID;
FilterEnum:= TSysDevEnum.Create(G);
```

On va chercher la chaîne de caractère correspondant au nom du périphérique tel qu'il est défini dans la liste dans ***GraphEdit***.
Il faut que le nom soit identique à celui-ci sinon l'ajout du filtre dans le graphe ne pourra pas se faire et vous n'obtiendrait pas le résultat escompté. C'est pour cette raison qu'il vaut mieux le sélectionner via une combobox que l'on a rempli avec la méthode de sélection que l'on a vu précédemment. En effet le programme teste le nom du filtre demandé par l'utilisateur avec la liste des filtres contenus dans la catégorie sélectionnée.

```
if FilterEnum.Filters[j].FriendlyName = sVideoRender then
```

Si le filtre existe on va obtenir son ID par l'intermédiaire de la ligne de code suivante :

```
CLSID_VideoRender:= FilterEnum.Filters[j].CLSID;
```

Il est important de pouvoir obtenir cet ID car par la suite on va s'en servir pour insérer le filtre dans le graphe par l'intermédiaire de la ligne de code suivante :

```
hr := CoCreateInstance(CLSID_VideoRender, nil, CLSCTX_INPROC_SERVER,
      IID_IBaseFilter, IVideoRenderFilter);
(fgRender as IGraphBuilder).AddFilter(IVideoRenderFilter, 'Pinnacle Video Render');
```

On va d'abord créer le filtre IVideoRender en utilisant la fonction COM CoCreateInstance à laquelle on va passer en paramètre l'ID du filtre que l'on a pu obtenir précédemment.
Il est à noter que si l'on a CLSID non valide on s'expose à obtenir une erreur de type :

Interface non supportée

Lorsque l'on va ajouter le filtre par l'intermédiaire de la fonction AddFilter.

d. Sélection et Ajout d'un filtre dans une liste

Dans cette partie nous allons voir comment il est possible de sélectionner des filtres dans la liste des filtres contenus sur votre ordinateur et de les placer dans une liste de type TStrings. Cette fonction va vous permettre de remplir une combobox par exemple.

```
function TfrmRenderer.GetSourcesCapture(SourceList : TStrings):boolean;
var i, j : integer;
   SysDev, FilterEnum:TSysDevEnum;
   g : TGUID;
begin
   result := False;
   try
   try

   SysDev:= TSysDevEnum.Create;

 for i:= 0 to SysDev.CountCategories-1  do
   begin
     if SysDev.Categories[i].FriendlyName = 'External Renderers' then
       begin
         SourceList.Add(SysDev.Categories[i].FriendlyName);

       end;
     if SysDev.Categories[i].FriendlyName = 'DirectShow Filters' then
       begin
       SourceList.Add(SysDev.Categories[i].FriendlyName);

       end;
   end;

   except
   end;
   finally
     SysDev.Free;
   end;

end;
```

Dans cet exemple nous allons sélectionner et ajouter dans une liste les Catégories « External Renderers » et « DirectShow Filters » si elles existent bien.

Pour pouvoir utiliser cette fonction vous pouvez taper la ligne de code suivante :

```
GetSourcesCapture(cbxVideoRendererCategory.Items);
```

Vous pourrez utiliser la fonction suivante pour ajouter dans une liste comportant les filtres tous les filtres correspondant à ces deux catégories :

```
cbxVideoRenderer.Clear;
   try
   try
```

```
   SysDev:= TSysDevEnum.Create;

 for i:= 0 to SysDev.CountCategories-1  do
   begin
    if SysDev.Categories[i].FriendlyName = cbxVideoRendererCategory.Text then
      begin

       G:= SysDev.Categories[i].CLSID;
       FilterEnum:= TSysDevEnum.Create(G);
       for j:= 0 to FilterEnum.CountFilters-1 do
         begin
          cbxVideoRenderer.Items.Add(FilterEnum.Filters[j].FriendlyName);
         end;
      end;

   end;
FilterEnum.Free;
   except
   end;
   finally
     SysDev.Free;
   end;
cbxVideoRenderer.Text := cbxVideoRenderer.Items[0];
```

En ce qui me concerne j'ai placé ce code dans la fonction OnChange de mon composant ***cbxVideoRendererCategory***. Comme cela lorsque je modifie le nom de ma catégorie la combobox permettant de sélectionner un filtre change en conséquence.

On commence par vider le contenu de la ***combobox cbxVideoRender***. Ensuite on applique la même méthode que précédemment pour ajouter les filtres à notre liste.

Ensuite il vous reste plus qu'à gérer le choix de votre filtre par l'intermédiaire de votre ***combobox cbxVideoRender***. Afin de le traiter en conséquence pour que vous puissiez ajouter votre filtre à votre graphe de construction comme vu précédemment.

e. Ajout d'un filtre intermédiaire et gestion du media

Il va souvent être nécessaire d'insérer des filtres intermédiaires tout simplement parce que DirectShow, même s'il possède un mode de création intelligent du graphe, peut ne pas arriver à finaliser celui-ci. Ou tout simplement parce que le périphérique a besoin d'informations spécifiques sur le média. Personnellement j'ai du utiliser cette technique pour diffuser des fichiers MPEG2 TS ***(.mts*** pour ***transfert stream***). Le filtre intermédiaire est un ***« Moonlight Elecard MPEG2 Demultiplexer »***.
Il s'insère en utilisant la même méthode que précédemment. Et voici le code obtenu :

```
SysDev:= TSysDevEnum.Create;

 for i:= 0 to SysDev.CountCategories-1  do
```

```
   begin
    if SysDev.Categories[i].FriendlyName = 'DirectShow Filters' then
      begin
        G:= SysDev.Categories[i].CLSID;
        FilterEnum:= TSysDevEnum.Create(G);
        for j:= 0 to FilterEnum.CountFilters-1 do
          begin
           if FilterEnum.Filters[j].FriendlyName = 'Moonlight-Elecard MPEG2
          Demultiplexer' then
             begin
               CLSID_IntermediateFilter:= FilterEnum.Filters[j].CLSID;
               break;
             end;
          end;
      end;

   end;
SysDev.Free;
FilterEnum.Free;

hr := CoCreateInstance(CLSID_IntermediateFilter, nil, CLSCTX_INPROC_SERVER,
      IID_IBaseFilter, IIntermediateFilter);
(fgRender as IGraphBuilder).AddFilter(IIntermediateFilter, 'Moonlight-Elecard MPEG2
Demultiplexer');
```

Ensuite lorsque ce filtre est inséré dans le graphe, on va lui adjoindre une structure de données comportant les données du media à diffuser :

```
ZeroMemory(@Mt, sizeof(AM_MEDIA_TYPE));
Mt.MajorType := MEDIATYPE_Video;
Mt.SubType := MEDIASUBTYPE_RGB32;
Mt.FormatType := FORMAT_MPEG2_VIDEO;
Mt.cbFormat := sizeof(MPEG2VIDEOINFO) + sizeof(segHdr);
mt.pbFormat := CoTaskMemAlloc(mt.cbFormat);
ZeroMemory(mt.pbFormat, mt.cbFormat);

RCSRC.Left := 0;
RCSRC.Top  := 0;
RCSRC.Right := 0;
RCSRC.Bottom := 0;
pWIH.hdr.rcSource := RCSRC;
pWIH.hdr.rcTarget := Rect(0,0,720,576);
pWIH.hdr.AvgTimePerFrame := 278335;
pWIH.hdr.dwPictAspectRatioX := 4;
pWIH.hdr.dwPictAspectRatioY := 3;
pWIH.hdr.bmiHeader.biSize := 40;
pWIH.hdr.bmiHeader.biWidth := 720;
pWIH.hdr.bmiHeader.biHeight := 480;
pWIH.cbSequenceHeader := sizeof(segHdr);
CopyMemory(@pwih.dwSequenceHeader, @segHdr, sizeof(seghdr));
```

Pour ce faire il faut d'abord déclarer la variable ***Mt*** de type ***AM_Media_Type*** :

Var mt : AM_Media_Type;

On alloue une zone mémoire à cette variable pour qu'elle pointe sur la structure de type AM_MEDIA_TYPE. On peut voir qu'il s'agit bien d'un clip vidéo de type MPE2, codé en image RGB 32 bits.

Les informations importantes sont ***MajorType***, ***SubType***, ***FormatType***. Les informations qui viennent ensuite sont des informations d'ordre général, sur la taille des images vidéo, les dimensions des images de sortie. Ces valeurs sont standard dans le monde de l'audiovisuel.

Ensuite il faut connecter le filtre intermédiaire au reste du graphe pour que ces informations soient prises en compte.

Ceci est effectué par les lignes de code suivantes :

```
SourceFilter.FindPin('Output',PinOutSource);
IAudioIntermediateFilter.FindPin('Input',PInIntermediateFilter);
PinOutSource.Connect(PInIntermediateFilter, @mt);
```

Pour ce faire on va essayer de trouver la broche de sortie de notre filtre source et de la connecter au filtre intermédiaire. A cette fin on utilise la fonction ***FindPin*** à laquelle on donne comme paramètre le nom de la broche de sortie, c'est-à-dire « Output » (tel qu'il apparaît visuellement dans ***GraphEdit***). On essaye ensuite de trouver l'entrée du filtre intermédiaire. Et on connecte les deux broches via la fonction ***Connect*** en associant la structure de donnée que l'on a défini précédemment.

f. Fonction de Diffusion

Il ne reste plus qu'à donner l'ordre de diffusion du media et ensuite le contrôler. Cette tâche est effectuée en tapant ces lignes de code.

```
hr := CoCreateInstance(CLSID_CaptureGraphBuilder2, nil,
      CLSCTX_INPROC_SERVER, IID_ICaptureGraphBuilder2,
     ICapGraph);

ICapGraph.SetFilterGraph(fgRender as IGraphBuilder);

hr := ICapGraph.RenderStream(nil,nil,
       SourceFilter, nil, IVideoRenderFilter);

hr := ICapGraph.RenderStream(nil,nil,
       SourceFilter, nil, IAudioRenderFilter);
```

Il faut tout d'abord créer un autre type d'objet de type ICaptureGraphBuilder2 :

Var ICapGraph : ICaptureGraphBuilder2;

C'est ce graphe qui va permettre la diffusion de la vidéo.
C'est pour cette raison qu'on lui associe ensuite le graphe de conception au graphe de diffusion par l'intermédiaire de la fonction ***SetFilterGraph***.

On peut voir ensuite de lignes de code contenant la fonction de diffusion de ***RenderStream***. Cette méthode est nécessaire. La première fonction permet de diffuser la vidéo alors que la seconde permet la diffusion de l'audio. Il faut toujours respecter cet ordre pour que le rendu soit efficace.
Cette fonction est aussi utilisée pour les fonctions de capture et de preview.
Dans notre cas il s'agit d'une diffusion les deux premiers paramètres sont donc positionnés à la valeur ***nil***.
Une fois que l'on a dit au graphe que l'on voulait diffuser le clip vidéo et avec quels moyens le faire il faut contrôler sa diffusion. Nous avions défini dans la partie « a. » les variables mc et mEvent, c'est le moment de nous en servir maintenant. Tout d'abord il faut mettre en lecture notre filtergraph de la manière suivante :

fgRender.Play;

Une fois que celui-ci est prêt à diffuser, il faut le mettre en lecture grâce à notre objet de type IMediaControl de la façon suivante:

mc.Run;

Il faut ensuite contrôler les événements liés à l'application durant la diffusion de notre vidéo cela est possible avec l'objet ***mEvent*** que nous avons défini.

Il faut lui associer la fonction ***WaitForCompletion***. Le problème de cette fonction est qu'on lui associe souvent le paramètre INFINTE de la façon suivante :

mEvent.WaitForCompletion(INFINITE, 0);

Paramétrée telle qu'elle celle-ci ne permet aucune interaction et oblige l'utilisateur à attendre la fin de la diffusion pour reprendre la main sur l'application.
Il faut pour éviter cela utiliser un Timer qui une fois activé va nous permettre de reprendre la main sur l'application à n'importe quel moment de la diffusion de la vidéo.

```
RenderTimer.enabled := True;
nCount := 0;

while nCount = 0 do
    begin
    if mEvent <> nil then
      begin
      mEvent.WaitForCompletion(fTime, nCount);
      Application.ProcessMessages;
      end
```

```
    else break;
    end;
if mc <>nil then
mc.stop ;
```

La fonction ***Application.ProcessMessages*** va nous permettre d'intercepter les messages issus de l'application et de les traiter.

```
procedure TfrmMediaPlayer.RenderTimerTimer(Sender: TObject);
var ftime : TDateTime;

begin

 fTime := Now-fElapsed;

end;
```

Cette méthode va nous permettre d'intercepter les messages jusqu'à la fin de la diffusion (***mc.Stop***) ou l'arrêt de la diffusion par un événement quelconque.

VI. CONCLUSION

Nous avons pu voir tout au long de ce tutoriel comment réaliser une application multimédia architecturée autour de DirectShow en utilisant le pack de composant DSPACK 234. Il vous a permis de vous introduire dans un univers qui permet d'effectuer des interfaces multimédias très riches. Il vous aura surtout permis de réaliser votre premier lecteur de fichier vidéo.
Bien entendu cela n'est qu'un avant goût mais il vous permettra, enfin, je l'espère de vous aider dans l'apprentissage de cette technologie.