3. Die Klasse

```
type
  PDB_Enum = ^TDB_Enumeration;
  TTDB_Enumeration = class(TDB_Enumeration)
  private
    P_Master : PMaster;
    EnumInfo : TEnumInfo;

    function EnumAdapter : HRESULT;
    function EnumCombos(Adapter : Integer)  : HRESULT;

  public
    constructor create(Master : PMaster);
    destructor  destroy; override;

    function Enumerate : HRESULT; override;
  end;
```

Eigentlich nichts weltbewegendes. Der Constructor erwartet einen Pointer auf eine TMaster Struktur der in die Private Variable P_Master verfrachtet wird. Zur Erinnerung, in dieser Struktur finden wir:

```
ID3D9         : IDirect3D9;
ID3DDEVICE    : IDirect3DDevice9;
PresParams    : TD3DPRESENTPARAMETERS;
EnumInfo      : TEnumInfo;
IsEnumerated  : BOOLEAN;
```

Wir haben ID3D9, ein Interface um D3D9 objekte zu erstellen, ID3DDEVICE ist das Interface für die Device, PresParams ist die Struktur die benötigt wird um eine Device zu erstellen, dann EnumInfo, dies ist die im ersten Teil dieses Tutorials beschriebene Struktur sowie IsEnumerated, ein Schalter der uns anzeigt ob schon mal enumeriert wurde.

Als einzig öffentliche Funktion finden wir Enumerate sowie die privaten Funktionen EnumAdapter und AnumCombos.

Enumerate dient uns als Ausgangspunkt, um zum ersten das Interface IDirect3D9 zu initialisieren und zum zweiten die Beiden Privaten Funktionen anzustossen.

3.1 Enumerate

1.Pointer auf das IDirect3D9 interface erzeugen.

```
If P_Master.ID3D9 := Direct3DCreate9(D3D_SDK_VERSION) <> D3D_OK then …
```

Mit Direct3DCreate9 bekommen wir unser Interface, der Parameter D3D_SDK_VERSION ist als einziger zulässig und sorgt dafür, dass die Header Files mit der dll die zur Laufzeit verwendet wird übereinstimmen. Sollte hier ein Fehler auftreten, beenden wir die Enumeration und verlassen die Funktion mit einer Fehlermeldung.

2. Aufruf alle Adapter zu enumerieren

if `EnumAdapter <> S_OK then …`

Wir rufen lediglich unsere private Funktion EnumAdapter auf. Wenn als Ergebnis ein Fehler auftritt, beenden wir die Enumeration ebenfalls.

3. Aufruf um alle Combos für die Adapter zu enumerieren

```
for i := 0 to EnumInfo.AnzAdapter -1 do …
```

Es soll Leute geben, die haben mehr al seine Grafikkarte im Rechner, daher wird in EnumAdapter die Anzahl der selbigen in unsere Struktur gespeichert. Für jeden Adapter also, rufen wir folgende Funktion auf:

```
If EnumCombos <> S_OK then …
```

Auch hier brechen wir ab, wenn etwas schief life.

4. Enumerationsergebnisse in Master speichern

```
P_Master.EnumInfo := EnumInfo;
```

Die Ergebnisse in die große Master Struktur packen ☺

```
P_Master.IsEnumerated := TRUE;
```

und noch schnell mitgeteilt, dass die Enumeration heute schon gelaufen ist.

5. IDirect3D9 interface wieder freigeben.

```
P_Master.ID3D9 := nil;
Result := S_OK;
```

Nicht vergessen unser Interface wieder freizugeben und die Enumeration mit einem freudigen alles ok zu verlassen.

3.2 EnumAdapter

Als erstes brauchen wir ein paar Variablen:

```
      AdapterLoop,
      FormatLoop,
      ModeLoop,
      ModeCount    : Integer;
      d3ddspmode   : D3DDISPLAYMODE;
```

AdapterLoop verwenden wir als Schleifenvariable um über alle Grafikkarten zu laufen. FormatLoop ist ebenfalls eine Schleifenvariable, hier kommt zum ersten mal eine Constante ins Spiel, **const AdapterFormats**, wir brauchen alle zulässigen Formate des FrontBuffers und lesen diese mit FormatLoop aus. Mit ModeLoop durchlaufen wir alle Modes die für das jeweilige FrontBuffers – Format gefunden wurden. ModeCount ist eine temporäre Variable in der die Anzahl der Modes die ein bestimmtes Format des FrontBuffers unterstützt zwischengespeichert werden. In d3ddspmode speichern wir unsere gefundenen DisplayModes zwischen.

Anzahl der Grafikkarten:

```
      EnumInfo.AnzAdapter := P_Master.ID3D9.GetAdapterCount;
```

Als erstes lassen wir uns die Anzahl der Adapter von unserem IDirect3D9 Interface geben und speichern diese in unserer Struktur. Wenn aus irgendwelchen Gründen keine Grafikkarte gefunde (erkannt) wird, brechen wir alles ab.

Array für die Graka(Adapter) auf richtige Länge bringen:

```
      SetLength(EnumInfo.Adapter, EnumInfo.AnzAdapter);
```

Wo Daten hinsollen, muss erst mal platz geschaffen werden, deshalb vergrößern wir unser Array „Adapter“ in „TEnumInfo“ und bringen es auf die benötigte Länge, in diesem Fall je nach Anzahl Adapter.

```
//Loop über alle Adapter(Grafikkarten)
for AdapterLoop := 0 to EnumInfo.AnzAdapter -1 do
  begin

   // Den Aktuellen DisplayMode speichern
   // Für den WindowedMode brauchen wir beim initialisieren der Device später diese
Angaben.
   // Wenn was schief geht, beenden.
   if P_Master.ID3D9.GetAdapterDisplayMode(AdapterLoop,
      EnumInfo.Adapter[AdapterLoop].DispModeSet) <> D3D_OK then
     begin
     Result := S_FALSE;
     exit;
     end;

   // Den Identifier der Grafikkarte (Adapter) speichern
   // DeviceName .. DriverVersion etc. sind  dort gespeichert
   if P_Master.ID3D9.GetAdapterIdentifier(AdapterLoop, 0,
      EnumInfo.Adapter[AdapterLoop].AdIdentif) <> D3D_OK then
     begin
     Result := S_FALSE;
     exit;
     end;
```

```
  //Loop über alle erlaubten Formate (const AdapterFormats = 3)
  // Wir haben 3 Zulässige Adapterformate in unserem Constanten Array [0..2]
  for FormatLoop := 0 to 2 do
    begin

    // Wieviele Modes werden mit diesem Format unterstützt
    ModeCount := P_Master.ID3D9.GetAdapterModeCount(AdapterLoop,
AdapterFormats[FormatLoop]);
    // Wenn keiner - ab zum nächsten
    if ModeCount <= 0 then continue;


    //Loop über Alle Modes
    for ModeLoop := 0 to ModeCount -1 do
      begin
      //Display Mode mit dem Format(AdapterFormats) enumerieren und in d3ddspmode
zwischenspeichern
      P_Master.ID3D9.EnumAdapterModes(AdapterLoop, AdapterFormats[FormatLoop],
                                      ModeLoop, d3ddspmode);

      // die zahl der Modes um eins erhöhen
      inc(EnumInfo.Adapter[AdapterLoop].AnzDispModes);

      // Array DispModes[] um eins vergrößern
      SetLength(EnumInfo.Adapter[AdapterLoop].DispModes,
                EnumInfo.Adapter[AdapterLoop].AnzDispModes);

       // Speichern von d3ddspmode in unsere Struktur
      ..[AdapterLoop].DispModes[..[AdapterLoop].AnzDispModes -1]:= d3ddspmode;

      end; // ** mode Loop
    end; // ** Format Loop
  end; // ** Adapter Loop

Result := S_OK;
```

Im Großen und Ganzen eine simple Angelegenheit.

3.2 EnumCombos

So, jetzt wird ein wenig aufwändiger, denn bedingt durch die Struktur werden Alle Aufrufe sehr lang, deshalb werde ich mich bemühen hier eine verständliche Kurzfassung zu liefern.

Als erstes erwartet die Funktion einen Integer Wert der den Adapter repräsentiert. Dies wird in Enumerate ja über die Schleife erledigt. (`for i := 0 to EnumInfo.AnzAdapter -1 do ...`)

Auch hier haben wir wieder einen ganzen Batzen an lokalen Variablen.

```
DeviceLoop,
AdptFmtLoop,
WndFuLoop,
BBFmtLoop,
DSLoop,
MSLoop        : Integer;
bWindowed     : BOOLEAN;
tmpDevice     : Array[0..2] of TDeviceInfo;
MSQuality     : CARDINAL;
AdptFmtSwitch : BOOLEAN;
```

Die Loop Variablen sind allesamt schleifenvariablen die, bis auf WndFuLoop, mit den Constanten Arrays zusammen funktionieren. WndFuLoop ist ein Schalter, der entweder 0 = Windowed Mode oder 1 = Fullscreen Mode annehmen kann - bWindowed repräsentiert eben diese Schleifenvariable als BOOLscher Wert für den Funktionsaufruf mit DX.
Damit unsere Codezeilen nicht absolut jeden Rahmen sprengen, legen wir ein Temporäres Array tmpDevice an und arbeiten innerhalb dieser Funktion damit.
MSQuality ist das QualityLevel das wir später als out Parameter von DX bekommen.
Den AdptFmtSwitch benötigen wir, um festzustellen ob ein neuer Speicherplatz im Array geschaffen werden soll oder ob wir uns noch im Schleifendurchlauf des BBFmtLoop befinden und keinen neuen Speicherplatz benötigen.

Loop über die **Devices(HAL/SW/REF)** des Adapters
- Wenn Device da, CAPS speichern – Die Caps unterscheiden sich je nach DeviceTyp
- Hardware Support vorhanden ?
- VertexProcessing festlegen – Wenn wir HardwareSupport haben, können wir auch DX-Light initialisieren
  und eine PureDevice erstellen. Wenn man allerdings umschalten möchte, und die Softwareemulation nutzen dann
  muss man das an dieser Stelle festlegen oder vor der Initialisierung der Device einen „Schalter“ einbauen der ein
  zurückschalten erlaubt.

Loop über alle zulässigen **Adapter-Formate**, hinterlegt in der Constanten AdapterFormats

Loop Fullscreen/Windowed (0 = Wondowed, 1 = Fullscreen)
- AdptFmtSwitch setzen, ist TRUE wenn eine neue Combo im Array angelegt werden muss
  Bei jedem neuen Switch in Windowed oder bei wechsel des Adapter Formats muss auch im Array später Ein neuer Speicherplatz angelegt werden.
- bWindowed setzen (ja nach 0 oder 1 des Loops)

Loop über alle **Backbuffer Formate**

- überprüfen ob eine Combo AdapterFormat<->Backbuffer<->Fullscreen/Windowed da ist, wenn nein: nächster Schleifendurchlauf.

```
ID3D9.CheckDeviceType(Adapter,
                      D3DDevTypes[DeviceLoop],
                      AdapterFormats[AdptFmtLoop],
                      BackBufferFmt[BBFmtLoop],
                      bWindowed) ;
```

Wenn hier <> D3D_OK, dann war es KEINE Combo.

- Wenn eine neue Combo, Platz im Array ComboInfo schaffen
- Platz im Array BBInfo schaffen und speichern

Loop über zulässige **Depth/Stencil** Formate mit const DepthStencilFmt

- Kompatibel mit dem DisplayFormat(fmt_Adapter)

```
CheckDeviceFormat
```

- Kompatibel mit dem Backbuffer

```
CheckDepthStencilMatch
```

Wenn einer dieser Beiden Tests fehlschlägt, ab zum nächsten Schleifendurchlauf.
Den Zähler Anz_fmt_Stencil um eins erhöhen
Array *StencilInfo[]* vergrössern
speichern des DepthStencil Formats in *StencilInfo[].fmt_Stencil*

Loop über **Multisamples** mit const MultiSampleTypes

- Multisample mit Backbuffer prüfen

```
CheckDeviceMultiSampleType(.. , .. , fmt_BackBuffer, .., .., ..)
```

- Multisample mit DepthStencil prüfen

```
CheckDeviceMultiSampleType(.. , .. , fmt_Stencil, .., .., ..)
```

- Quality Level abfragen

```
CheckDeviceMultiSampleType(.. , .. , fmt_BackBuffer, .., .., @MSQuality)
```

- Platz im Array *MSInfo[]* schaffen
- Speichern

```
MSQuality := MSQuality-1;
```

MSQuality wird mit -1 gespeichert, weil der aufruf bei DX Index -1 ist ☺

```
MSTyp := MultiSampleTypes[MSLoop]
```

- Zähler eins rauf

```
Anz_MSInfos +1
```

```
        end; //MSLoop
      end; //DSLoop
    end; // Loop BacBufferFormate mit const BackBufferFmt
  end; // Loop Fullscren Windowed
 end; // Loop AdapterFormate mit const AdapterFormats//
end; // Loop Hal .. SW .. REF
```

Speichern der temporären DeviceInfos in enumInfo

```
      enumInfo.Adapter[Adapter].DeviceInfo[0] := tmpDevice[0];
      enumInfo.Adapter[Adapter].DeviceInfo[1] := tmpDevice[1];
      enumInfo.Adapter[Adapter].DeviceInfo[2] := tmpDevice[2];
```

Wir sind angekomme und verabschieden uns aus dieser Funktion mit einem alles ok.

```
Result := S_OK;
```

Damit haben wir alles, was unsere Grafikkarte beherrscht enumeriert und an die richtige Stelle in unserer Struktur gepackt. Wir müssen, auf der Suche nach einem bestimmten Setting für die Initialisierung der Device, lediglich die entsprechenden Arrays durchlaufen und mit den vorgaben vergleichen. Eine Funktion um die Engine zu starten könnte dann z.B. so aussehen:

```
function Init3DFullscreen (Adpt : INTEGER; Width, Height, RR, AdptFmtBits, DepthBits
                           SINGLE; Stencil : BOOLEAN; MSLevel : Integer;
                           FormHandle : HWND) : HRESULT; override;
```

Und als Aufruf:

```
Engine.Init3DFullscreen(0,800,600, 0, 32,16,FALSE,4, Form1.Handle)
```

01.02.2005
MexDelphi