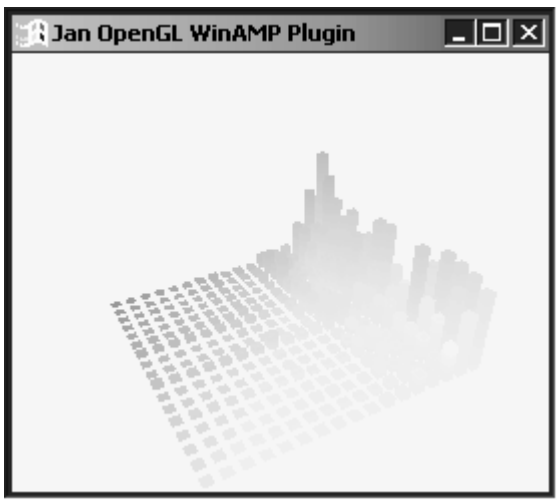
فصل بیست و دوم

ایجاد برنامه های افزودنی سه بعدی برای WinAMP



مقدمه :

بدون شک یکی از بهترین ها در دنیای پخش فایل های MP3 نرم افزار محبوب WinAMP می باشد . همراه با این برنامه تعداد بی شماری فایل های افزودنی (PLUG-IN) نیز وجود دارد که جلوه های گوناگونی را به آن اضافه می کنند . در این فصل طریقه ایجاد Plugins سه بعدی برای WinAMP را خواهید آموخت .

مروری بر رکوردها و توابع مورد نیاز :

یک Plugin فایلی است DLL (با فرمتی که توضیح داده خواهد شد) که پس از تولید باید در دایرکتوری WinAMP\Plugins کپی شود . سپس در محیط WinAMP دکمه های Ctrl+K را بفشارید تا انواع مختلف برنامه های افزودنی WinAMP را مشاهده و اجرا نمایید .
برای ساخت یک Plugin در محیط دلفی ابتدا یک پروژه ساخت DLL را باز کنید . فرض WinAMP بر این است که DLL شما تابع زیر را در اختیار آن قرار می دهد :

```
exports winampVisGetHeader;
```

سپس رکوردهای زیر را به برنامه خود اضافه کنید . WinAMP از آنها برای اجرای توابع مورد نیازش استفاده می کند :

PwinampVisModule & PwinampVisHeader

```
type
  PWinampVisModule = ^TwinampVisModule;
  TwinampVisModule = record
    description  : PChar;        // description of module
    hwndParent   : HWND;         // parent window (filled in by calling app)
    hDllInstance : HINST;        // instance handle to this DLL (filled in by calling app)
    sRate        : Cardinal;     // sample rate (filled in by calling app)
    nCh          : Cardinal;     // number of channels (filled in...)
    latencyMs    : Cardinal;     // latency from call to Render to actual drawing
    delayMs      : Cardinal;     // delay between calls to Render (in ms)

    // the data is filled in according to the respective Nch entry
    spectrumNCh  : Cardinal;     // Number of channels
    waveformNCh  : Cardinal;     // Number of channels
    spectrumData : Array [0..1, 0..575] of Byte;     // waveform data   (values from 0-255)
    waveformData : Array [0..1, 0..575] of Byte;     // spectrum data   (values from 0-255)

    // functions that winamp calls to configure the plugin, initialise ...
    Config       : procedure(const PVisModule : PwinampVisModule); cdecl;
    Init         : function (const PVisModule : PwinampVisModule) : Integer; cdecl;
    Render       : function (const PVisModule : PwinampVisModule) : Integer; cdecl;
    Quit         : procedure(const PVisModule : PwinampVisModule); cdecl;
    userData     : procedure; cdecl;  // user data, optional
  end;

  PwinampVisHeader = ^TwinampVisHeader;
  TwinampVisHeader = record
    version      : Integer;
    description  : PChar;  // description of library
    getModule    : function (Which : Integer) : PwinampVisModule; cdecl;
  end;
```

توابع Callback که توسط WinAMP فراخوانی خواهند شد ، در قسمت بعد آورده شده اند :

```
// forward declaration of the procedures
  function  GetModule(Which :integer) :PWinAMPVisModule; cdecl;
  procedure Config(const PVisModule : PWinAMPVisModule); cdecl;
  function  Init(const   PVisModule : PWinAMPVisModule) : integer; cdecl;
  function  Render(const PVisModule : PWinAMPVisModule) :integer; cdecl;
  procedure Quit(const   PVisModule : PWinAMPVisModule); cdecl;
```

تابع خروجی زیر را نیز قبل از شروع کردن به نوشتن اصل برنامه ، اضافه نمایید :

```
function winampVisGetHeader : PwinampVisHeader; cdecl; export;
```

متغیرهای عمومی مورد استفاده :

قسمت زیر بسته به سلیقه شما قابل تغییر است :

```
implementation

const
  WND_TITLE = 'Jan OpenGL WinAMP Plugin';
var
  h_Wnd  : HWND;                  // Global window handle
  h_DC   : HDC;                  // Global device context
  h_RC   : HGLRC;                 // OpenGL rendering context
  keys : Array[0..255] of Boolean;   // Holds keystrokes
  Active : Boolean = FALSE;          // State of the plugin
```

توابع که توسط WinAMP فراخوانی می شوند :

: WinAMPVisGetHeader

اشاره گری را به WinAMPVisHeader بر می گرداند . متغیری حاوی اطلاعات Plugin و توابع آن .

```
function WinAMPVisGetHeader :PWinAMPVisHeader;
begin
  Result := @HDR;    //  Return the main header
end;
```

تابع به فرمت زبان دلفی	تابع به فرمت زبان C
function GetModule(Which :integer) :PWinAMPVisModule; cdecl;	winampVisModule* (*getModule)(int);

اشاره گری را به WinampVisModule بر می گرداند . متغیری حاوی اطلاعات Plugin و توابع آن .

```
function GetModule(Which : integer) : PwinampVisModule;
begin
  if which = 0 then
    Result := @VisModule
```

```
  else
    Result := nil;
end;
```

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void (*Config)(struct winampVisModule *this_mod); // configuration dialog	procedure Config(const PVisModule : PWinAMPVisModule); cdecl;

این تابع هنگامی فراخوانی می گردد که کاربر در قسمت تنظیمات Plugin را در WinAMP انتخاب می کند . تنظیمات Plugin مانند اندازه صفحه ، محل قرارگیری آن و غیره می باشد و سپس در فایل plugin.ini در دایرکتوری WinAMP\Plugins ذخیره می گردد .

```
procedure Config(const PVisModule : PWinAMPVisModule);
begin
  Form1 :=TForm1.Create(nil);
  try
    Form1.ShowModal;
  finally
    Form1.Free;
  end;
end;
```

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
int (*Init)(struct winampVisModule *this_mod); // 0 on success, creates window, etc	function Init(const PVisModule : PWinAMPVisModule) : integer; cdecl;

تابع آغازگر که اطلاعات مربوط به تنظیمات ذخیره شده در فایل Plugin.ini را برای نمایش دریافت می کند . تمام متغیرهای عمومی در اینجا مقدار دهی اولیه می شوند .

```
function Init(const PVisModule :PWinAMPVisModule) :integer;
begin
  ...
  // Get plugin config details
  ...
  // create and initialise the window
  ...
end;
```

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
int (*Render)(struct winampVisModule *this_mod); // returns 0 if successful, 1 if vis should end	function Render(const PVisModule : PWinAMPVisModule) : integer; cdecl;

اصلی ترین تابع است که تمام اتفاقات در آن رخ می دهد . تعداد دفعاتی که این تـابع فراخـوانی می شود وابسته است به مقدار DelayMS در VisModule . برای مثال مقدار ۲۵ به WinAMP مـی گویـد که این تابع را هر ۲۵ میلی ثانیـه یکبـار اجـرا نمایـد . در اینجـا رنـدر کـردن تصـاویر و در یافـت اطلاعات مربوط به فشرده شدن کلیدها انجام می شود .

```
function Render(const PVisModule : PWinAMPVisModule) : Integer;
var LastTime : DWord;
begin
  if Active then
  begin
    glDraw(PVisModule);              // Draw the scene
    SwapBuffers(h_DC);               // Display the scene

    if (keys[VK_ESCAPE]) then        // If user pressed ESC then set finised TRUE
    begin
      Active :=FALSE;
      PostQuitMessage(0);
      Result :=1;
      exit;
    end
    else
      ProcessKeys(PVisModule);       // Check for any other key Pressed
  end;
  Result :=0;
end;
```

تابع به فرمت زبان دلفی	تابع به فرمت زبان C
procedure Quit(const PVisModule : PWinAMPVisModule); cdecl;	void (*Quit)(struct winampVisModule *this_mod); // call when done

پنجره را می بندد .

```
procedure Quit(const PVisModule : PWinAMPVisModule);
begin
  glKillWnd(PVisModule, AppFullScreen);
end;
```

اگر مایل به دریافت اطلاعات بیشتری در زمینه نوع هـای دیگـر برنامـه هـای افزودنـی WinAMP هستید به آدرس زیر مراجعه نمایید :

http://www.winamp.com/nsdn/winamp2x/dev/plugins/

برنامه فصل :

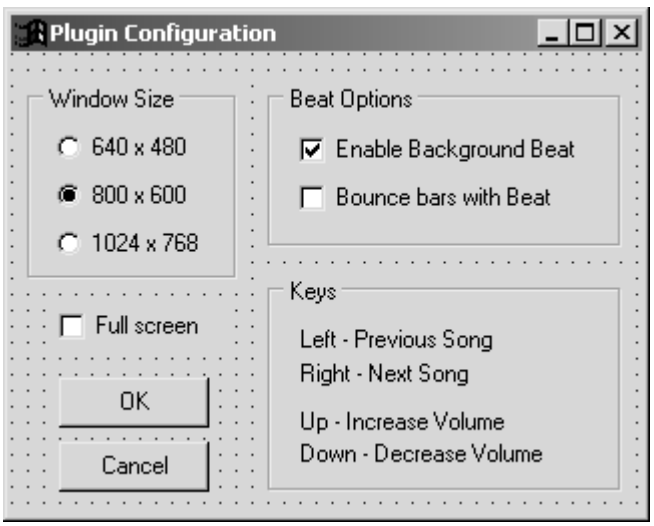
در برنامه زیر از توابع API ویندوز برای ایجاد فرم اصلی برنامه کمک گرفته شده است که بصورت قابل توجهی حجم فایل EXE حجیم دلفی را کاهش می دهد .

ابتدا یک پروژه ساخت DLL را باز کنید و سپس کد زیر را در آن بنویسید :

```
// Description : winamp visualization plugin
//
//----------------------------------------------------------------------
library DelphiOpenGL;
uses
  Windows,
  Glplugin ,  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}
  exports winampVisGetHeader;
end.
```

سپس یک فرم به برنامه اضافه کنید و کد مربوط به آنرا به شرح زیر وارد نمایید :




```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
```

```
    RadioButton3: TRadioButton;
    CheckBox1: TCheckBox;
    GroupBox2: TGroupBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    GroupBox3: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
implementation
Uses IniFiles;
{$R *.DFM}
{-----------------------------------------------------------------}
{--- Form create : Load settings from INI file. if there are  ---}
{---   no settings, the default is assumed                    ---}
{-----------------------------------------------------------------}
procedure TForm1.FormCreate(Sender: TObject);
var PluginIni : TIniFile;
    Path : String;
    P : Integer;
    Width : Integer;
begin
  // Get path to winamp plugin.ini file
  Path :=ParamStr(0);   // eg. 'c:\program files\winamp\winamp.exe'
  P :=Length(Path);
  while Path[P] <> '\' do
    Dec(P);
  Path :=Copy(Path, 1, P);
  Path :=Path + 'Plugins\';

  PluginIni := TIniFile.Create(Path + 'plugin.ini');

  // Get window settings
  Width :=PluginIni.ReadInteger('JansGL', 'Width', 800);
  if Width = 640 then
    RadioButton1.Checked :=TRUE
  else if Width = 800 then
    RadioButton2.Checked :=TRUE
  else if Width = 1024 then
    RadioButton3.Checked :=TRUE;

  CheckBox1.Checked :=PluginIni.ReadBool('JansGL', 'FullScreen', FALSE);

  // Get display settings
```

```
  CheckBox2.Checked :=PluginIni.ReadBool('JansGL', 'BackgroundBeat', TRUE);
  CheckBox3.Checked :=PluginIni.ReadBool('JansGL', 'BounceBeat', FALSE);

  PluginIni.Free;
end;

procedure TForm1.Button1Click(Sender: TObject);
var PluginIni : TIniFile;
    Path : String;
    P : Integer;
begin
  // Get path to winamp plugin.ini file
  Path :=ParamStr(0);   // eg. 'c:\program files\winamp\winamp.exe'
  P :=Length(Path);
  while Path[P] <> '\' do
    Dec(P);
  Path :=Copy(Path, 1, P);
  Path :=Path + 'Plugins\';
  PluginIni := TIniFile.Create(Path + 'plugin.ini');

  // Save the window size
  if RadioButton1.Checked then
  begin
    PluginIni.WriteInteger('JansGL', 'Width', 640);
    PluginIni.WriteInteger('JansGL', 'Height', 480);
  end;
  if RadioButton2.Checked then
  begin
    PluginIni.WriteInteger('JansGL', 'Width', 800);
    PluginIni.WriteInteger('JansGL', 'Height', 600);
  end;
  if RadioButton3.Checked then
  begin
    PluginIni.WriteInteger('JansGL', 'Width', 1024);
    PluginIni.WriteInteger('JansGL', 'Height', 768);
  end;
  // save fullscreen information
  PluginIni.WriteBool('JansGL', 'FullScreen', CheckBox1.Checked);

  // save background beat information
  PluginIni.WriteBool('JansGL', 'BackgroundBeat', CheckBox2.Checked);

  // save bounce beat information
  PluginIni.WriteBool('JansGL', 'BounceBeat', CheckBox3.Checked);

  PluginIni.Free;

  Close;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Close;
end;


end.
```

و نهایتا unit GLplugin که آنرا در فایل اصلی پروژه مشاهده نمودید به شرح زیر می باشد :

```
unit GLplugin;

interface

uses Windows, Messages, OpenGL, Unit1;
procedure glGenTextures(n: GLsizei; var textures: GLuint); stdcall; external opengl32;
procedure glBindTexture(target: GLenum; texture: GLuint); stdcall; external opengl32;
// WinAmp type declarations from vis.h
type
  PWinampVisModule = ^TwinampVisModule;
  TwinampVisModule = record
    description  : PChar;        // description of module
    hwndParent   : HWND;         // parent window (filled in by calling app)
    hDllInstance : HINST;        // instance handle to this DLL (filled in by calling app)
    sRate        : Cardinal;     // sample rate (filled in by calling app)
    nCh          : Cardinal;     // number of channels (filled in...)
    latencyMs    : Cardinal;     // latency from call of RenderFrame to actual drawing\
                          // (calling app looks at this value when getting data)
    delayMs      : Cardinal;     // delay between calls in ms

    // the data is filled in according to the respective Nch entry
    spectrumNCh  : Cardinal;     // Number of channels
    waveformNCh  : Cardinal;     // Number of channels
    spectrumData : Array [0..1, 0..575] of Byte;     // waveform data   (values from 0-255)
    waveformData : Array [0..1, 0..575] of Byte;     // spectrum data   (values from 0-255)
    Config       : procedure(const PVisModule : PwinampVisModule); cdecl;
   // configuration dialog
    Init         : function (const PVisModule : PwinampVisModule) : Integer; cdecl;
    // 0 on success, creates window, etc
    Render       : function (const PVisModule : PwinampVisModule) : Integer; cdecl;
   // returns 0 if successful, 1 if vis should end
    Quit         : procedure(const PVisModule : PwinampVisModule); cdecl;
    // call when done
    userData     : procedure; cdecl;  // user data, optional
  end;
  PwinampVisHeader = ^TwinampVisHeader;
  TwinampVisHeader = record
    version      : Integer;
    description  : PChar;  // description of library
    getModule    : function (Which : Integer) : PwinampVisModule; cdecl;
  end;

  // forward declaration of the procedures
  function  GetModule(Which :integer) :PWinAMPVisModule; cdecl;
  procedure Config(const PVisModule : PWinAMPVisModule); cdecl;
  function  Init(const   PVisModule : PWinAMPVisModule) : integer; cdecl;
  function  Render(const PVisModule : PWinAMPVisModule) :integer; cdecl;
  procedure Quit(const   PVisModule : PWinAMPVisModule); cdecl;

// Winamp TWinAMPVisHeader
const
  HDR : TWinAMPVisHeader =
```

```
        (Version      : $101;  // WinAMP checks this for compatibility
         Description  : 'Jan OpenGL WinAMP Plugin';
         GetModule    : GetModule);
  VisModule : TWinAMPVisModule =
        (Description  :'OpenGL Spectrum Analyzer';
         hWNDParent   :0;
         hDLLInstance :0;
         sRate        :0;
         nCh          :0;
         LatencyMs    :10;
         DelayMS      :33;
         SpectrumNch  :2;
         WaveformNch  :0;
         Config       :Config;    // config function
         Init         :Init;      // Plug-in initialization function. Returns 0 if success
         Render       :Render;    // render function. 0 if successful, 1 if vis should end
         Quit         :Quit;      // quit function
         UserData     :nil);


function winampVisGetHeader : PwinampVisHeader; cdecl; export;

implementation
Uses IniFiles;
const
  WND_TITLE = 'Jan OpenGL WinAMP Plugin';
var
  h_Wnd  : HWND;                  // Global window handle
  h_DC   : HDC;                   // Global device context
  h_RC   : HGLRC;                 // OpenGL rendering context
  keys : Array[0..255] of Boolean;   // Holds keystrokes
  Active : Boolean = FALSE;

  PluginStart : DWord;
  ElapsedTime : DWord;            // Elapsed time between frames
  AppFullScreen : Boolean;
  EnableBeat : Boolean;
  BounceBeat : Boolean;
  VUBars : Array[0..17, 1..17] of glFloat;   // 18 VU bars with a history of 17 steps

{------------------------------------------------------------------}
{  Draws a VU bar. Bars go to a lighter blue to the right and      }
{  fade to black as we go backwards                                }
{------------------------------------------------------------------}
procedure VUBox(X, Y, Z, Value : glFloat);
var C, F : glFloat;
    R, G, B : glFloat;
begin
  C :=(11.75 + X)/55;  // amount of color to fade by as we go to the right
  F :=(11.5-Z)/60;     // amount of fade to black as we go back

  R :=C-F;            // RED
  G :=C-F;            // GREEN
  B :=1-F;            // BLUE
  // front
  glColor3f(R, G, B);
    glVertex3f(X, Y, Z);
    glVertex3f(X+0.9, Y, Z);
```

```
  glColor3f(Value/10+R, G, B-Value/10);   // VUBars = values 0 - 10
    glVertex3f(X+0.9, Y+Value, Z);
    glVertex3f(X,   Y+Value, Z);
  // left
  glColor3f(R, G, B);
    glVertex3f(X, Y, Z-0.9);
    glVertex3f(X, Y, Z);
  glColor3f(Value/10+R, G, B-Value/10);
    glVertex3f(X, Y+Value, Z);
    glVertex3f(X, Y+Value, Z-0.9);

  // back
  glColor3f(R, G, B);
    glVertex3f(X+0.9, Y, Z-0.9);
    glVertex3f(X,   Y, Z-0.9);
  glColor3f(Value/10+R, G, B-Value/10);
    glVertex3f(X,   Y+Value, Z-0.9);
    glVertex3f(X+0.9, Y+Value, Z-0.9);
  // right
  glColor3f(R, G, B);
    glVertex3f(X+0.9, Y, Z);
    glVertex3f(X+0.9, Y, Z-0.9);
  glColor3f(Value/10+R, G, B-Value/10);
    glVertex3f(X+0.9, Y+Value, Z-0.9);
    glVertex3f(X+0.9, Y+Value, Z);
  // top
  glColor3f(R, G, B);
    glVertex3f(X, Y, Z);
    glVertex3f(X+0.9, Y, Z);
  glColor3f(Value/10+R, G, B-Value/10);
    glVertex3f(X+0.9, Y+Value, Z-0.9);
    glVertex3f(X, Y+Value, Z-0.9);
end;

{--------------------------------------------------------------------}
{  Function to draw the actual scene                                 }
{--------------------------------------------------------------------}
procedure glDraw(const PVisModule : PWinAMPVisModule);
var I, J : Integer;
    BackgroundBeat : glFloat;
    X, Y, Z : glFloat;
begin
  // BackGround Beat Color - Average the first 10 values of SpectrumData
  BackGroundBeat :=0;
  For I:=0 to 9 do
    BackGroundBeat :=BackgroundBeat + PVisModule^.spectrumData[0][I] +
PVisModule^.spectrumData[1][I];
  BackgroundBeat :=BackgroundBeat /10 /255 ;

  if EnableBeat then
    glClearColor(0, 0, BackgroundBeat/2, 0)
  else
    glClearColor(0, 0, 0, 0);


  // Clear screen and setup coordinates
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  // Clear The Screen And The Depth Buffer
```

```
  glLoadIdentity();                              // Reset The View
  glTranslatef(0.0, -5.0, -50);
  glRotate(30, 1, 0, 0 );
  glRotate(ElapsedTime/100, 0, 1, 0);

  // Translate spectrumData array values to OpenGL metrics
  // first shift all the bars back one row.
  For J :=16 downto 1 do
    For I :=0 to 17 do
      VUBars[I, J+1] :=VUBars[I, J];

  // zero first row
  For I :=0 to 17 do
    VUBars[I, 1] :=0;

  // average all left and right 288 VU bars into 18 VU bars.
  For I :=0 to 287 do                {left}                    {right}
    VUBars[I DIV 16, 1] := VUBars[I DIV 16, 1] + PVisModule^.spectrumData[0][I] +
PVisModule^.spectrumData[1,I];

  // trim the extra high peaks
  For I :=0 to 17 do
  begin
    VUBars[I, 1] :=VUBars[I, 1]/51/2;
    if VUBars[I, 1] > 15 then
      VUBars[I, 1] :=15;
  end;

  // display the VU bars
  if BounceBeat then
    Y :=BackGroundBeat
  else
    Y :=0;

  glBegin(GL_QUADS);
    For J := 1 to 17 do
    begin
      Z :=13-J*1.5;
      For I :=1 to 17 do
      begin
        X :=-13.25 + I*1.5;
        VUBox(X, Y, Z, VUBars[i, j]);
      end;
    end;
  glEnd();
end;

{------------------------------------------------------------------}
{  Initialise OpenGL                                               }
{------------------------------------------------------------------}
function glInit : Boolean;
begin
  glClearColor(0.0, 0.0, 0.0, 0.0);            // Black Background
  glShadeModel(GL_SMOOTH);                     // Enables Smooth Color Shading
  glClearDepth(1.0);                   // Depth Buffer Setup
  glEnable(GL_DEPTH_TEST);                     // Enable Depth Buffer
  glDepthFunc(GL_LESS);                                // The Type Of Depth Test To Do
```

```
  glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
//Realy Nice perspective calculations
  Result :=TRUE;
end;

{-----------------------------------------------------------------}
{  WINAMP function - Get plugin handle                            }
{-----------------------------------------------------------------}
function WinAMPVisGetHeader :PWinAMPVisHeader;
begin
  Result := @HDR;    //  Return the main header
end;


{-----------------------------------------------------------------}
{  WINAMP function - Get module handle                            }
{-----------------------------------------------------------------}
function GetModule(Which : integer) : PwinampVisModule;
begin
  if which = 0 then
    Result := @VisModule
  else
    Result := nil;
end;

{-----------------------------------------------------------------}
{  WINAMP function - Config setting for the vis module            }
{-----------------------------------------------------------------}
procedure Config(const PVisModule : PWinAMPVisModule);
begin
  Form1 :=TForm1.Create(nil);
  try
    Form1.ShowModal;
  finally
    Form1.Free;
  end;
end;

{-----------------------------------------------------------------}
{  Handle window resize                                           }
{-----------------------------------------------------------------}
procedure glResizeWnd(Width, Height : Integer);
begin
  if (Height = 0) then              // prevent divide by zero exception
    Height := 1;
  glViewport(0, 0, Width, Height);    // Set the viewport for the OpenGL window
  glMatrixMode(GL_PROJECTION);        // Change Matrix Mode to Projection
  glLoadIdentity();                   // Reset View
  gluPerspective(45.0, Width/Height, 1.0, 100.0);
 // Do the perspective calculations. Last value = max clipping depth

  glMatrixMode(GL_MODELVIEW);         // Return to the modelview matrix
  glLoadIdentity();                   // Reset View
end;

{-----------------------------------------------------------------}
{  Processes all the keystrokes                                   }
{-----------------------------------------------------------------}
```

```
procedure ProcessKeys(const PVisModule : PWinAMPVisModule);
begin
  // Jump To Prev Song
  if keys[VK_LEFT] then
     SendMessage(PVisModule^.hWNDParent, WM_COMMAND, 40044, 0);

  // Jump To Next Song
  if keys[VK_RIGHT] then
     SendMessage(PVisModule^.hWNDParent, WM_COMMAND, 40048, 0);

  // Toggle Volume Up
  if keys[VK_UP] then
     SendMessage(PVisModule^.hWNDParent, WM_COMMAND, 40058, 0);

  // Toggle Volume Down
  if keys[VK_DOWN] then
     SendMessage(PVisModule^.hWNDParent, WM_COMMAND, 40059, 0);

//  if keys[VK_F3] then PeakFallOff:=PeakFallOff -0.003;
//  if keys[VK_F4] then PeakFallOff:=PeakFallOff +0.004;

//if keys[VK_F5] then if Flash then Flash:=False else Flash:=True;

end;

{------------------------------------------------------------------}
{  Determines the application's response to the messages received  }
{------------------------------------------------------------------}
function WndProc(hWnd: HWND; Msg: UINT;  wParam: WPARAM;  lParam: LPARAM): LRESULT;
stdcall;
begin
  case (Msg) of
    WM_CREATE:
      begin
        // Insert stuff you want executed when the program starts
      end;
    WM_CLOSE:
      begin
        PostQuitMessage(0);
        Result := 0
      end;
    WM_KEYDOWN:
     // Set the pressed key (wparam) to equal true so we can check if its pressed
      begin
        keys[wParam] := True;
        Result := 0;
      end;
    WM_KEYUP:
     // Set the released key (wparam) to equal false so we can check if its pressed
      begin
        keys[wParam] := False;
        Result := 0;
      end;
    WM_SIZE:          // Resize the window with the new width and height
      begin
        glResizeWnd(LOWORD(lParam),HIWORD(lParam));
        Result := 0;
      end;
```

```
    else
      Result := DefWindowProc(hWnd, Msg, wParam, lParam);
     // Default result if nothing happens
  end;
end;

{------------------------------------------------------------------}
{  Properly destroys the window created at startup (no memory leaks)  }
{------------------------------------------------------------------}
procedure glKillWnd(const PVisModule : PWinAMPVisModule; Fullscreen : Boolean);
begin
  if Fullscreen then            // Change back to non fullscreen
  begin
    ChangeDisplaySettings(devmode(nil^), 0);
    ShowCursor(True);
  end;

  // Makes current rendering context not current, and releases the device
  // context that is used by the rendering context.
  if (not wglMakeCurrent(h_DC, 0)) then
    MessageBox(0, 'Release of DC and RC failed!', 'Error', MB_OK or MB_ICONERROR);

  // Attempts to delete the rendering context
  if (not wglDeleteContext(h_RC)) then
  begin
    MessageBox(0, 'Release of rendering context failed!', 'Error', MB_OK or MB_ICONERROR);
    h_RC := 0;
  end;

  // Attemps to release the device context
  if ((h_DC = 1) and (ReleaseDC(h_Wnd, h_DC) <> 0)) then
  begin
    MessageBox(0, 'Release of device context failed!', 'Error', MB_OK or MB_ICONERROR);
    h_DC := 0;
  end;

  // Attempts to destroy the window
  if ((h_Wnd <> 0) and (not DestroyWindow(h_Wnd))) then
  begin
    MessageBox(0, 'Unable to destroy window!', 'Error', MB_OK or MB_ICONERROR);
    h_Wnd := 0;
  end;

  // Attempts to unregister the window class
  if (not UnRegisterClass('OpenGL',PVisModule^.hDllInstance)) then
  begin
    MessageBox(0, 'Unable to unregister window class!', 'Error', MB_OK or MB_ICONERROR);
    hInstance := 0;
  end;
end;

{------------------------------------------------------------------}
{  WINAMP function - Config setting for the vis module            }
{------------------------------------------------------------------}
procedure Quit(const PVisModule : PWinAMPVisModule);
begin
  glKillWnd(PVisModule, AppFullScreen);
end;
```

```
{------------------------------------------------------------------}
{  Creates the window and attaches a OpenGL rendering context to it  }
{------------------------------------------------------------------}
function glCreateWnd(const PVisModule : PWinAMPVisModule;
          Width, Height, PixelDepth : Integer; Fullscreen : Boolean) : Boolean;
var
  wndClass : TWndClass;         // Window class
  dwStyle : DWORD;              // Window styles
  dwExStyle : DWORD;            // Extended window styles
  dmScreenSettings : DEVMODE;   // Screen settings (fullscreen, etc...)
  pfd : TPIXELFORMATDESCRIPTOR;  // Settings for the OpenGL window
  PixelFormat : glUint;
begin
  ZeroMemory(@wndClass, SizeOf(wndClass));  // Clear the window class structure

  with wndClass do                  // Set up the window class
  begin
    style         := CS_HREDRAW or    // Redraws entire window if length changes
                     CS_VREDRAW or    // Redraws entire window if height changes
                     CS_OWNDC;        // Unique device context for the window
    cbClsExtra:=0;                  // No Extra Window Data
    cbWndExtra:=0;                  // No Extra Window Data
    lpfnWndProc   := @WndProc;        // Set the window procedure to our func WndProc
    hInstance     := PVisModule^.hDllInstance;
    hbrBackground := 0;               // No BackGround Required For OpenGL
    lpszMenuName  := nil;             // We Don't Want A Menu
    hCursor       := 0;
    lpszClassName := 'OpenGL';
  end;

  if (RegisterClass(wndClass) = 0) then  // Attemp to register the window class
  begin
    MessageBox(0, 'Failed to register the window class!', 'Error', MB_OK or MB_ICONERROR);
    Result := False;
    Exit
  end;

  // Change to fullscreen if so desired
  if Fullscreen then
  begin
    ZeroMemory(@dmScreenSettings, SizeOf(dmScreenSettings));
    with dmScreenSettings do begin              // Set parameters for the screen setting
      dmSize       := SizeOf(dmScreenSettings);
      dmPelsWidth  := Width;                  // Window width
      dmPelsHeight := Height;                 // Window height
      dmBitsPerPel := PixelDepth;              // Window color depth
      dmFields     := DM_PELSWIDTH or DM_PELSHEIGHT or DM_BITSPERPEL;
    end;

    // Try to change screen mode to fullscreen
    if (ChangeDisplaySettings(dmScreenSettings, CDS_FULLSCREEN) = DISP_CHANGE_FAILED) then
    begin
      MessageBox(0, 'Unable to switch to fullscreen!', 'Error', MB_OK or MB_ICONERROR);
      Fullscreen := False;
    end;
  end;
```

```
// If we are still in fullscreen then
if (Fullscreen) then
begin
  dwStyle := WS_POPUP or              // Creates a popup window
          WS_CLIPCHILDREN             // Doesn't draw within child windows
          or WS_CLIPSIBLINGS;         // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW;       // Top level window
  ShowCursor(False);                  // Turn of the cursor (gets in the way)
end
else
begin
  dwStyle := WS_OVERLAPPEDWINDOW or   // Creates an overlapping window
          WS_CLIPCHILDREN or          // Doesn't draw within child windows
          WS_CLIPSIBLINGS;            // Doesn't draw within sibling windows
  dwExStyle := WS_EX_APPWINDOW or     // Top level window
            WS_EX_WINDOWEDGE;         // Border with a raised edge
end;

// Attempt to create the actual window
h_Wnd := CreateWindowEx(dwExStyle,    // Extended window styles
                  'OpenGL',           // Class name
                  WND_TITLE,          // Window title (caption)
                  dwStyle,            // Window styles
                  0, 0,               // Window position
                  Width, Height,      // Size of window
                  0,                  // No parent window
                  0,                  // No menu
                  PVisModule^.hDllInstance,
                  nil);               // Pass nothing to WM_CREATE
if h_Wnd = 0 then
begin
  glKillWnd(PVisModule, Fullscreen);              // Undo all the settings we've changed
  MessageBox(0, 'Unable to create window!', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
end;

// Try to get a device context
h_DC := GetDC(h_Wnd);
if (h_DC = 0) then
begin
  glKillWnd(PVisModule, Fullscreen);
  MessageBox(0, 'Unable to get a device context!', 'Error', MB_OK or MB_ICONERROR);
  Result := False;
  Exit;
end;

// Settings for the OpenGL window
with pfd do
begin
  nSize          := SizeOf(TPIXELFORMATDESCRIPTOR); // Size Of This Pixel Format Descriptor
  nVersion       := 1;                  // The version of this data structure
  dwFlags        := PFD_DRAW_TO_WINDOW  // Buffer supports drawing to window
                 or PFD_SUPPORT_OPENGL // Buffer supports OpenGL drawing
                 or PFD_DOUBLEBUFFER;  // Supports double buffering
  iPixelType     := PFD_TYPE_RGBA;      // RGBA color format
  cColorBits     := PixelDepth;         // OpenGL color depth
  cRedBits       := 0;                  // Number of red bitplanes
```

```
    cRedShift      := 0;               // Shift count for red bitplanes
    cGreenBits     := 0;               // Number of green bitplanes
    cGreenShift    := 0;               // Shift count for green bitplanes
    cBlueBits      := 0;               // Number of blue bitplanes
    cBlueShift     := 0;               // Shift count for blue bitplanes
    cAlphaBits     := 0;               // Not supported
    cAlphaShift    := 0;               // Not supported
    cAccumBits     := 0;               // No accumulation buffer
    cAccumRedBits  := 0;               // Number of red bits in a-buffer
    cAccumGreenBits := 0;              // Number of green bits in a-buffer
    cAccumBlueBits := 0;               // Number of blue bits in a-buffer
    cAccumAlphaBits := 0;              // Number of alpha bits in a-buffer
    cDepthBits     := 16;              // Specifies the depth of the depth buffer
    cStencilBits   := 0;               // Turn off stencil buffer
    cAuxBuffers    := 0;               // Not supported
    iLayerType     := PFD_MAIN_PLANE;      // Ignored
    bReserved      := 0;               // Number of overlay and underlay planes
    dwLayerMask    := 0;               // Ignored
    dwVisibleMask  := 0;               // Transparent color of underlay plane
    dwDamageMask   := 0;               // Ignored
  end;

  // Attempts to find the pixel format supported by a device
// context that is the best match to a given pixel format specification.
  PixelFormat := ChoosePixelFormat(h_DC, @pfd);
  if (PixelFormat = 0) then
  begin
    glKillWnd(PVisModule, Fullscreen);
    MessageBox(0, 'Unable to find a suitable pixel format', 'Error', MB_OK or MB_ICONERROR);
    Result := False;
    Exit;
  end;

  // Sets the specified device context's pixel format to
//the format specified by the PixelFormat.
  if (not SetPixelFormat(h_DC, PixelFormat, @pfd)) then
  begin
    glKillWnd(PVisModule, Fullscreen);
    MessageBox(0, 'Unable to set the pixel format', 'Error', MB_OK or MB_ICONERROR);
    Result := False;
    Exit;
  end;

  // Create a OpenGL rendering context
  h_RC := wglCreateContext(h_DC);
  if (h_RC = 0) then
  begin
    glKillWnd(PVisModule, Fullscreen);
    MessageBox(0, 'Unable to create an OpenGL rendering context', 'Error', MB_OK or
MB_ICONERROR);
    Result := False;
    Exit;
  end;

  // Makes the specified OpenGL rendering context the calling
// thread's current rendering context
  if (not wglMakeCurrent(h_DC, h_RC)) then
  begin
```

```
    glKillWnd(PVisModule, Fullscreen);
    MessageBox(0, 'Unable to activate OpenGL rendering context', 'Error', MB_OK or MB_ICONERROR);
    Result := False;
    Exit;
  end;

  // Settings to ensure that the window is the topmost window
  ShowWindow(h_Wnd, SW_SHOW);
  SetForegroundWindow(h_Wnd);
  SetFocus(h_Wnd);

  // Ensure the OpenGL window is resized properly
  glResizeWnd(Width, Height);
  if not glInit then
  begin
    glKillWnd(PVisModule, Fullscreen);
    MessageBox(0,'OpenGL Initialization failed.','Error',MB_OK or MB_ICONEXCLAMATION);
    Result := false;       //Return False
    exit;
  end
  else
    Result := true;       //Return False
end;


{--------------------------------------------------------------------}
{  Plugin Init section. Gets the plugin saved details from the      }
{  plugin.ini file in winamp\plugins and creates the GL window      }
{--------------------------------------------------------------------}
function Init(const PVisModule :PWinAMPVisModule) :integer;
var Width, Height : Integer;
    PluginIni : TIniFile;
    Path : String;
    P : Integer;
begin
  // Get path to winamp plugin.ini file
  Path :=ParamStr(0);   // eg. 'c:\program files\winamp\winamp.exe'
  P :=Length(Path);
  while Path[P] <> '\' do
    Dec(P);
  Path :=Copy(Path, 1, P);
  Path :=Path + 'Plugins\';

  PluginIni := TIniFile.Create(Path + 'plugin.ini');

  // Get window settings
  Width :=PluginIni.ReadInteger('JansGL', 'Width', 800);
  Height:=PluginIni.ReadInteger('JansGL', 'Height', 600);
  AppFullScreen :=PluginIni.ReadBool('JansGL', 'FullScreen', FALSE);

  // Get display settings
  EnableBeat :=PluginIni.ReadBool('JansGL', 'BackgroundBeat', TRUE);
  BounceBeat :=PluginIni.ReadBool('JansGL', 'BounceBeat', FALSE);

  PluginIni.Free;

  if not glCreateWnd(PVisModule, Width, Height, 32, AppFullScreen) then
  begin
```

```
    glKillWnd(PVisModule, AppFullScreen);
    Result := 1;    //Quit If The Window Wasn't Created
    exit;
  end;

  PluginStart := GetTickCount();          // Get Time when demo started

  Result := 0;
  Active :=TRUE;
end;

{------------------------------------------------------------------}
{  Main message loop for the application                           }
{------------------------------------------------------------------}
function Render(const PVisModule : PWinAMPVisModule) : Integer;
var LastTime : DWord;
begin
  if Active then
  begin
    LastTime :=ElapsedTime;
    ElapsedTime :=GetTickCount() - PluginStart;     // Calculate Elapsed Time
    ElapsedTime :=(LastTime + ElapsedTime) DIV 2;
   // Average it out for smoother movement

    glDraw(PVisModule);               // Draw the scene
    SwapBuffers(h_DC);                // Display the scene

    if (keys[VK_ESCAPE]) then          // If user pressed ESC then set finised TRUE
    begin
      Active :=FALSE;
      PostQuitMessage(0);
      Result :=1;
      exit;
    end
    else
      ProcessKeys(PVisModule);         // Check for any other key Pressed
  end;
  Result :=0;
end;

end.
```