Displaying video files using Free Pascal and Lazarus

Michaël Van Canneyt

November 18, 2012

Abstract

In a recent contribution, it was shown how video can be recorded with Lazarus on windows. In this contribution, we show how to display arbitrary video files using lazarus on Windows and Linux.

1 Introduction

Many people know VideoLan as an excellent open source cross-platform media player: it plays an enormous amount of audio and video formats, and does this on many platforms: at least Windows, Linux and Mac OS. It can be downloaded from

```
http://www.videolan.org/
```

Less people know that the core functionality of the VideoLan player is available in a library, which is distributed with the player, and which the player itself uses.

This library (there are actually 2 libraries) can be used in other programs to display video files. The window in which the video is displayed can be embedded in another window, managed by the program that uses the library. This allows you to develop your own front-end to the videolan player.

The header files of the videolan library (called libvlc) have been translated to pascal, and a thin OOP layer was constructed around it. This OOP layer matches closely the object model introduced by Videolan. Using the three components in the layer allows one to play videolan files without any GUI at all: the videos will be displayed in a free-floating window, or even full screen.

From the main component in the VLC OOP layer, a descendent was made that can be used in Lazarus: it introduces only the capability of embedding the video window in a lazarus control. (a similar component exists for fpGUI, an alternative to the LCL maintained by Graeme Geldenhuys).

All code has been checked in to the Subversion source archives of Free Pascal and Lazarus, but only in the development version (trunk). It is expected to be added to one of the next releases. For readers that want to try it, all relevant code is available for download as well.

Lazarus users can install the lazvlc package. It will install 2 components on the component palette: A playlist manager and a media player component.

Although everything was created for Free Pascal and Lazarus, the code should be compilable (possibly with minor adjustments) with Delphi as well.

2 Architecture

There are 3 units that must be used when using the VLC libraries:

libvlc This unit contains the low-level translation of the libvlc header files (vlc.h). Functions from this unit should normally not be used, but it contains also some constants and enumerated types that must be used. The unit

vlc This unit contains the OOP layer around the low-level library. It contains 5 classes, described below.

lclvlc This unit contains the LCL version of the VLC player component: `TLCLVLCPlayer`. If the lazvlc package is installed, then the component is registered on the component palette.

The vlc unit contains several classes:

TVLCLibrary This class represents the loaded VLC library. This component is a wrapper around the VLC `libvlc_instance_t` type. It can be used to set the library location and global options of the library, and has some properties and methods that allow to get the version of the library.

TVLCMediaPlayer this is the media player: the component that actually shows the video files. It contains the Play, Pause, Resume and Stop methods one expects from a video player component. The component is a wrapper around the VLC `libvlc_media_player_t` type.

TVLCMediaItem This class (a `TCollectionItem` descendent) represents 1 video or audio file or stream that can be played by the `TVLCMediaPlayer` class. Its main property is the `Path`, indicating the file that needs to be played. It has also an URL property or a FileDescriptor property for remote media. This collection item class is a wrapper around the `libvlc_media_t` type in Libvlc.

TVLCMediaItems is the collection corresponding to the `TVLCMediaItem` item. It wraps the `libvlc_media_list_t` type.

TVLCMediaListPlayer is a play list manager component. The play list is in a `TVLCMediaItems` collection, and an instance of `TVLCMediaPlayer` is used to actually play the items in the collection. It wraps the VLC `libvlc_media_list_player_t` type.

From the description, it is clear that for each type introduced in the VLC library, there is a corresponding pascal class.

3 Loading and initializing the library

The VLC library is loaded on demand, and after loading, it must be initialized. This process can be controlled through the `TVLCLibrary` component.

Under normal circumstances, no instance of the `TVLCLibrary` must be created. The `VLCLibrary` function gives access to a global instance of this class. It will dynamically load the VLC library as soon as its `Initialize` method is called. Normally, there should be no need to call this method manually: it will be called as soon as one of the other classes is used.

The `LibraryPath` property can be set to the name and location of the VLC library. If set, the system assumes that the core library is in the same location. If not set, the system will

assume a default name and location. The `LibraryArgs` stringlist can be used to pass a series of arguments to the library. These arguments are the same command-line arguments that the VLC player itself uses.

Care must be taken when initializing the library. There are 2 points to look out for:

1. The library loads some video drivers. Some of these drivers cause floating point errors, which are converted to exceptions. These exceptions can be disabled with the `SetExceptionMask` function of the Math unit, as follows:

   ```
   setexceptionmask([exInvalidOp, exDenormalized, exZeroDivide,
                     exOverflow, exUnderflow, exPrecision]);
   ```

   The exceptions can be enabled again with

   ```
   setexceptionmask([]);
   ```

   Not doing this will disable all exceptions caused by floating point math errors.

2. The VLC library uses threads. Since these threads are created outside of the Free Pascal RTL scope, the RTL does not know about them, and the threading system is not initialized. This causes errors in case VLC callbacks (event handlers) are used; These handlers are called from the context of a thread created by VLC. if the program that uses the VLC library does not use threads by itself, the threading system must be initialized anyway. This can be done with the following statement:

   ```
   With TThread.Create(False) do
     Terminate;
   ```

On windows, the VideoLan installation sets a registry key with the location of the VLC installation. This can be used to set the correct library path in case VLC was not installed in the default location, as follows:

```
Procedure SetVLCLibraryPath;

Const
  VLCKey = 'Software\VideoLAN\VLC';
var
  D : String;

begin
  Result := '';
  With TRegistry.Create(KEY_READ) do
    try
      RootKey:=HKEY_LOCAL_MACHINE;
      if OpenKey(VLCKey,False) then
        if ValueExists('InstallDir') then
          begin
          D:=r.ReadString('InstallDir'));
          D:=IncludeTrailingPathDelimiter(D);
          VLCLibrary.LibraryPath:=D+libvlc.libname;
          end;
    finally
      Free;
    end;
  end;
end;
```

4 A simple commandline player

After the VLC library was initialized, the `TVLCMediaPlayer` component can be used to play a video or audio file.

This class has the following important methods:

```
procedure SetMedia(M: TVLCMediaItem);
procedure Play;
Procedure Play(M : TVLCMediaItem);
Procedure PlayFile(Const AFileName : String);
Procedure Stop;
procedure Pause;
procedure Resume;
procedure NextFrame;
function Snapshot(Const AFileName: String): Boolean;
function Snapshot(Const AFileName: String;
                  AWidth, AHeight: Cardinal): Boolean;
function GetVideoSize(Var AWidth, AHeight: Cardinal): Boolean;
```

The meaning of these functions is almost self explaining:

SetMedia Prepares a media item for playing.

Play Plays a media item prepared with SetMedia. If a media item is passed to this function, then it is prepaed using Setmedia, end then played.

PlayFile will play a file: it creates a TVLCMediaItem, initializes it with the given filename, and then calls play.

Stop stops the player from playing the current media.

Pause pauses the player during playback.

Resume resumes playing if the player was paused.

NextFrame Jumps to the next frame in the video.

GetVideoSize Returns the width and height of the currently playing video.

Snapshot will create a snapshot from the current video frame and saves it to the specified file. If no width and height are specified, then they are determined using `GetVideoSize`.

Note that the control commands such as play, resume work asynchronously: they return immediatly. That is because VLC performs the work in background threads. The current state of the VLC player can be examined with the followin properties:

Playing A boolean indicatin whether the player is currently playing a media file ?

State Gives more detailed information about the current status.

The status can be one of the following:

libvlc_NothingSpecial The player is idle.

libvlc_Opening The player is opening a media file or stream.

libvlc_Buffering The player is buffering data for playback.

libvlc_Playing The player is playing a media item.

libvlc_Paused The player was paused with `Pause`.

libvlc_Stopped The player was stopped with the `Stop` command.

libvlc_Ended The player reached the end of the media file.

libvlc_Error The player encountered an error from which it could not recover.

Armed with these properties and commands, a minimal command-line program that plays a media file can be constructed as follows:

```
program testvlc;

{$mode objfpc}{$H+}

uses
  {$ifdef unix}cthreads,{$endif}
  sysutils, math, libvlc, vlc;

Const
  AllExceptions =
   [exInvalidOp, exDenormalized, exZeroDivide,
    exOverflow, exUnderflow, exPrecision];

begin
  SetExceptionMask(AllExceptions);
  With TVLCMediaPlayer.Create(Nil) do
    try
      PlayFile(ParamStr(1));
      Repeat
        Sleep(100);
      until State in [libvlc_Ended,libvlc_Error];
    finally
      Free;
    end;
end.
```

The program is extremely simple. a `TVLCMediaPlayer` player instance is created, and the first command-line argument is passed on to it with `PlayFile`. Then a loop starts that waits till the player has stopped playing. The player instance is freed, and the program exits. It is hard to get more simple than that.

5 Managing a playlist

The `TVLCPlayer` component plays one media item at a time. The `TVLCMediaItem` can be part of a collection of media items. The VLC library has functionality to maintain and play a collection in a playlist. This functionality is wrapped in the `TVLCMediaListPlayer`. It has the same playing commands (Play, Pause, Stop) as the `TVLCMediaPlayer` component, but has additionally the `Prev` and `Next` commands to jump to the next or previous items in the playlist.

In addition, it has the following properties:

Player This points to an instance of the `TVLCMediaPlayer` component that should be used to play the media items.

PlayMode This enumeration property controls how the items are played: this is one of `pmNormal`, `pmLoop`, `pmRepeat`: their meaning is quite clear.

MediaItems This is the collection of `TVLCMediaItems` that is being played.

Using this component, the command-line program can be extended to play a list of files instead of just one file:

```
program testvlc2;

{$mode objfpc}{$H+}

uses
  {$ifdef unix}cthreads,{$endif}
  sysutils, math, libvlc, vlc;

Const
  AllExceptions =
   [exInvalidOp, exDenormalized, exZeroDivide,
    exOverflow, exUnderflow, exPrecision];

Var
  i : integer;

begin
  SetExceptionMask(AllExceptions);
  with TVLCMediaListPlayer.Create(Nil) do
    try
      Player:=TVLCMediaPlayer.Create(Nil);
      For I:=1 to ParamCount do
        TVLCMediaItem(MediaItems.Add).Path:=ParamStr(i);
      Play;
      Repeat
        Sleep(100);
      until State in [libvlc_Ended,libvlc_Error];
    finally
      Player.Free;
      Free;
    end;
end.
```

Setting the `Path` property of a `TVLCMediaItem` collection item is all that is needed to play a file.

6 Using additional properties and events in a GUI

The commands and examples that were given until now do not demonstrate any interaction with the player: a file is passed, and the program waits till the player has finished playing. This is of course not very interesting, and in a real-world application, the user will want some more control over the player.

There are several properties that can be used to examine or even influence the currently playing media file. The first pr

AudioTrackCount The number of audio tracks in the video file. Read-only.

AudioTrackDescriptions A zero-based array of strings, describing the audio tracks (the language). Read-only.

AudioTrack The audio track (zero-based) to use when playing the video (read/write).

AudioDelay A delay (in milliseconds) to use when playing audio (read/write).

AudioVolume The audio volume. An integer number between 0 and 200 (read/write).

AudioMuted A boolean which can be set to mute the sound (read/write).

Channel The audio channel being used.

The following properties give some information about the video:

ChapterCount The number of chapters in the video. Read-only.

Chapter The chapter being played currently. Read/Write.

VideoWidth The video width, in pixels. Read-only.

VideoHeight The video height, in pixels. Read-only.

VideoLength The video length, in milliseconds. Read-only.

VideoDuration The video length as a date/time value. Read-only.

VideoPosition The current position in the video file, in milliseconds. Read/write.

VideoFractionalPosition The current position in the video file, as a percent of the full length. Read/Write

VideoFramesPerSecond The number of frames per second. Read-only.

VideoScale The scale of the video, where 1 is normal.

AspectRatio A string decribing the aspect ratio. Read/write.

Last but not least, the following properties can be set to determine how the video is played on screen:

FullScreenMode If set to true, the video is played full-screen.

FitWindow If set to true, then the parent video window will be resized to fit the video size.

UseEvents if set to true, then the libVLC callbacks will be installed, and a number of event handlers will be activated.

There are many event handlers available:

OnMediaChanged Occurs when the media changes for the player. Mostly relevant when used with a playlist.

OnNothingSpecial Occurs when the player is idle.

OnBackward Called when a video is seeking backwards.

OnBuffering Called when the player is buffering data for playback.

OnEOF Called when the end of a media is reached.

OnError Called when libvlc encounters an error.

OnForward Called when a video is seeking forwards.

OnOpening Called when a media is opened.

OnPause Called when playback is paused.

OnPlaying Called when playing resumes.

OnStop Called when playback is stopped.

OnLengthChanged Called when the length of the video changes. The new length is passed on as a parameter. This is usually called only once when the length has been determined.

OnTimeChanged Called when the current time in the video has changed. The current time is passed on.

OnPausableChanged Called when the ability to pause the video changed. This is called normally only once, at the beginning of the video.

OnPositionChanged Called when the current position in the video has changed. The current position is passed on.

OnSeekableChanged Called when the ability to seek the video has changed. This is called normally only once, at the beginning of the video.

OnTitleChanged Called when the current title changes.

OnSnapshot Called when a snapshot was made.

The events are called from the videolan thread that is playing the video. This means that if the events are used to update the display, e.g. set a trackbar to the correct position, or display the current and total times, then the `Synchronize` procedure must be used to update the display.

Last but not least, the LCL version of the control has a property `ParentWindow`. This can be set to point to any `TWinControl` descendent, and the video will be played embedded in this control. Normally, one would use a `TPanel` for this.

To demonstrate some of these properties, a small video player can be constructed. It will have a menu bar, toolbar, a panel in which the video is shown, and a panel with 2 trackbars: one for sound, one for the position in the video.

The toolbar and menu will have buttons to open a file, and to control playback: play, resume, pause. All buttons and menu items are controlled using actions.

In the OnCreate event of the form, the player is created, and the events are hooked up:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FPlayer:=TLCLVLCPlayer.Create(Self);
  FPlayer.ParentWindow:=PVideo;
  FPlayer.OnTimeChanged:=@DoTimeChanged;
  FPlayer.OnPositionChanged:=@DoPositionChanged;
  FPlayer.OnLengthChanged:=@DoLengthChanged;
  FPlayer.UseEvents:=True;
end;
```

The panel in which the video is shown is caled `PVideo`. There will be 3 events that we use: Two events to display the time and length of the video, and one to set the trackbar to the current position in the video.

Playing a video (the AOpen action) is very simple:

```
 procedure TMainForm.MIOpenClick(Sender: TObject);
begin
  With ODVideo do
   begin
   FileName:=FFileName;
   if Execute then
     begin
     FFileName:=FileName;
     FPlayer.PlayFile(FFileName);
     Caption:='Lazarus video demo: '+FFIleName;
     end;
   end;
end;
```

To update the display while the video is playing, 3 events have been implement. When VLC reports a change of position in `OnPositionChanged`, the `TBVideo` trackbar's position is updated as follows:

```
procedure TMainForm.DoPositionChanged(Sender: TObject; const APos: Double);
begin
  FNewPosition:=Round(APos*100);
  TThread.Synchronize(Nil,@SetNewPosition);
end;
```

It saves the new position: the fractional value is converted to a value between 1 and 100. Since the event handler is called in a thread, the form must be updated in a synchronized method, `SetNewPosition`:

```
procedure TMainForm.SetNewPosition;

begin
  FShowing:=True;
  try
    TBVideo.Position:=FNewPosition;
  finally
    FShowing:=False;
  end;
end;
```

The variable `FShowing` is used to notify the `OnChange` event handler of the trackbar that the position is being set from the video. This way, the handler can decide whether it is the user changing the position in the video, or whether it is the video player that is playing and reporting its new position:

```
procedure TMainForm.TBVideoChange(Sender: TObject);
begin
  if not FShowing then
    FPlayer.VideoFractionalPosition:=TBVideo.Position/100;
end;
```

Note that the position must be converted to a fractional position (0=start, 1=end of video).

Similarly, the total and current times are reported with the `OnTimeChanged` and `OnLengthChanged` event handlers:

```
procedure TMainForm.DoTimeChanged(Sender: TObject;
                                  const time: TDateTime);
begin
  FCurrentTime:=Time;
  TThread.Synchronize(Nil,@DisplayTime);
end;

procedure TMainForm.DoLengthChanged(Sender: TObject;
                                    const time: TDateTime);
begin
  FNewLength:=Time;
  TThread.Synchronize(Nil,@DisplayTime);
end;
```

Both event handlers will update the display in the `DisplayTime` routine:

```
procedure TMainForm.DisplayTime;

  Function TtoS (T : TDateTime) : string;
  Var
    h,m,s,ms : Word;
  begin
    DecodeTime(T,h,m,s,ms);
    if h>0 then
      Result:=FormatDateTime('hh:nn:ss',T)
    else
      Result:=FormatDateTime('nn:ss',T);
  end;
Var
  s : string;

begin
  S:='/';
  if FNewLength>0 then
    S:=S+TToS(FNewLength)
  else
    S:=S+'?';
  if (FCurrentTime>0) then
    S:=TToS(FCurrentTime)+S
  else
    S:='0:0'+S;
  LTime.Caption:=S;
end;
```

This routine does nothing other than trying to show a nice time/length in a label.

The `OnExecute` handlers of the Stop, Pause and Resume actions are straightforward:

```
procedure TMainForm.BStopClick(Sender: TObject);
begin
```

```
  FPlayer.Stop;
end;

procedure TMainForm.BPauseClick(Sender: TObject);
begin
  FPLayer.Pause;
end;

procedure TMainForm.BResumeClick(Sender: TObject);
begin
  FPlayer.Resume;
end;
```

And their `OnUpdate` handlers make use of the `State` property of the video player component:

```
procedure TMainForm.AStopUpdate(Sender: TObject);
begin
  (Sender as TAction).Enabled:=FPlayer.Playing;
end;

procedure TMainForm.APauseUpdate(Sender: TObject);
begin
  (Sender as TAction).Enabled:=FPlayer.Playing;
end;

procedure TMainForm.AResumeUpdate(Sender: TObject);
begin
  (Sender as TAction).Enabled:=FPlayer.State=libvlc_Paused;
end;
```

Remains to implement the volume control through a trackbar (`TBVolume`):

```
procedure TMainForm.TBVolumeChange(Sender: TObject);
begin
  FPlayer.AudioVolume:=TBVolume.Position;
end;
```

With all this, a functional video player is ready to go. The result can be seen in figure 1 on page 12.

7 Conclusion

The VideoLan libraries are a very good choice if you want to embed a video player in your cross-platform application: Videolan supports a large amount of video formats, can play streams from other computers, and has an easy-to-use API which has been mapped in a quite natural way to Object Pascal classes. Its includion in the Free Pascal and Lazarus distributions makes it probably the easiest way to play video in a cross-platform Lazarus application.

Figure 1: The video player in action