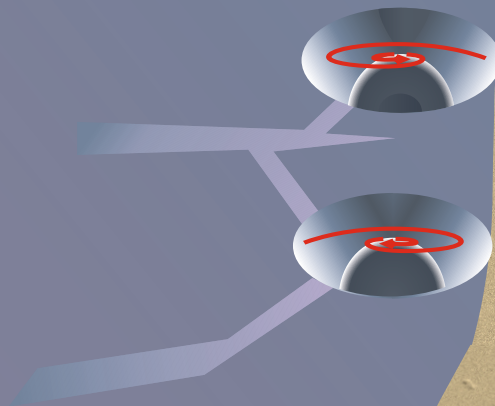


F O R D E L P H I, L A Z A R U S, A N D P A S C A L
R E L A T E D L A N G U A G E S / W E B A P P S,
I N T E R N E T, A N D R O I D, I O S, M A C,
W I N D O W S & L I N U X



Blaise Pascal

BLAISE PASCAL MAGAZINE 73/74



Quantum Internet: Professor Stephanie Wehner

REST easy with kbmMW #14 – DB Controlled login: By Kim Madsen

Delphi revelations #1 – kbmMW Smart client on NextGen (Android) – Scope problems:

By Kim Madsen

kbmMW safety first #2 – Hardware based random numbers

By Kim Madsen

Inter-thread communication By Jean Pierre Hoefnagel

Creating Games Using Castle Game Engine: By Michalis Kamburelis

Video Processing:

Video capture, Screen capture, IP Camera, Webstreaming, Creating videos from Frames

By Boian Mitov

Rotation Button: By David Dirkse

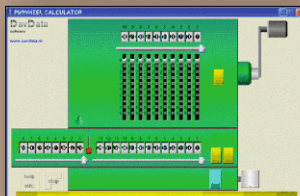
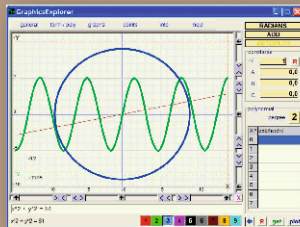
STATICS: Calculating loads that are hanging on a cable

By David Dirkse

Converting Delphi code to Lazarus

By Juha Manninen and Howard Page-Clark

DAVID DIRKSE



```
procedure ;
var
begin
  for i := 1 to 9
  do
    begin
```

GRAPHICS COMPUTER MATH & GAMES IN PASCAL

LIBRARY 2018



BLAISE PASCAL MAGAZINE

ALL ISSUES IN ONE FILE

POCKET EDITION

Printed in full color.
A fully indexed PDF book
is included + 52 projects

Editor in Chief: Detlef Overbeek
Edelstenenbaan 21 3402 XA
Usselstein Netherlands



Prof Dr.Wirth, Creator of Pascal Programming language

CREDITCARD LIBRARY STICK 16 GB

All issues 1-70 on the USB stick complete
searchable 3800 pages -fully indexed
including all code

BLAISE PASCAL MAGAZINE

```
procedure
var
begin
  for i := 1 to 9
  do
    begin
```

Prof Dr.Wirth, Creator of Pascal Programming language



Blaise Pascal, Mathematician

BLAISE PASCAL MAGAZINE

```
procedure
var
begin
  for i := 1 to 9
  do
    begin
```

Prof Dr.Wirth, Creator of Pascal Programming language



Blaise Pascal, Mathematician

COMBINATION: 3 FOR 1

BOOK INCLUDING THE LIBRARY STICK EXCL. SHIPPING
INCLUDING 1YEAR DOWNLOAD FOR FREE
GET THE BOOK INCLUDING THE NEWEST LIBRARY STICK
INCLUDING 1 YEAR DOWNLOAD OF BLAISE PASCAL MAGAZINE

€ 100

<https://www.blaisepascalmagazine.eu/product-category/special-offer/>

BLAISE PASCAL MAGAZINE 73

DELPHI, LAZARUS, SMARTMOBILE STUDIOS
AND PASCAL RELATED LANGUAGES
FOR ANDROID, IOS, MAC, WINDOWS & LINUX



Blaise Pascal

CONTENT

ARTICLES

Quantum Internet: Professor Stephanie Wehner

Page 30

REST easy with kbmMW #14 – DB Controlled login: By Kim Madsen

Page 22

Delphi revelations #1 – kbmMW Smart client on NextGen (Android)

Page 27

– Scope problems:

By Kim Madsen

kbmMW safety first #2 – Hardware based random numbers

Page 28

By Kim Madsen

Inter-thread communication By Jean Pierre Hoefnagel

Page 35

Creating Games Using Castle Game Engine: By Michalis Kamburelis

Page 47

Video Processing:

Page 82

Video capture, Scen capture, IP Camera, Webstreaming, Creating videos from Frames

By Boian Mitov

Rotation Button: By David Dirkse

Page 67

STATICS: Calculating loads that are hanging on a cable

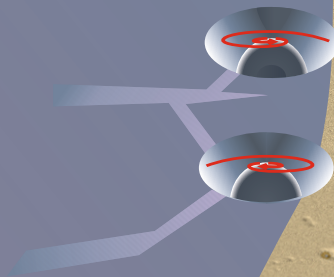
Page 10

By David Dirkse

Converting Delphi code to Lazarus

Page 72

By Juha Manninen and Howard Page-Clark



About the illustration: **QuTech:** We have succeeded in generating quantum entanglement between two quantum chips faster than the entanglement is lost. Entanglement – once referred to by Einstein as “spooky action” – forms the link that will provide a future quantum internet its power and fundamental security. Via a novel smart entanglement protocol and careful protection of the entanglement, we were able to deliver such a quantum link ‘on demand’. This opens the door to connect multiple quantum nodes and create the very first quantum network in the world. The results were published on 14 June in Nature.

ADVERTISERS

TMS Software Webcore

Page 6

TMS Software FNC

Page 7

TMS Software Busines Subscription

Page 8

Lazarus 2.0 on the event

Page 21

Pascon

Page 45/46

Lazarus Professional

Page 69/70/71

Mitov Software

Page 80

Components 4 Developers

Page 104



Pascal is an imperative and procedural programming language, which Niklaus Wirth designed in 1968–69 and published in 1970, as a small, efficient language intended to encourage good programming practices using structured programming and data structuring. A derivative known as Object Pascal designed for object-oriented programming was developed in 1985. The language name was chosen to honour the Mathematician, Inventor of the first calculator: Blaise Pascal (see top right).

Left: Niklaus Wirth



Stephen Ball http://delphiaball.co.uk @DelphiABall	Miguel Bebensee mbebensee@ibexpert.biz http://devstructor.com	Peter Bijlsma -Editor peter @ blaise-pascal.eu
Dmitry Boyarintsev dmitry.living @ gmail.com	Michaël Van Canneyt, michael @ freepascal.org	Marco Cantù www.marcocantu.com marco.cantu @ gmail.com
David Dirkse www.davdata.nl E-mail: David @ davdata.nl	Benno Evers b.evers @ evercustomtechnology.nl	Bruno Fierens www.tmssoftware.com bruno.fierens @ tmssoftware.com
Holger Flick holger@flixments.com		
Primož Gabrijelčič www.primoz @ gabrijelcic.org	Mattias Gärtner nc-gaertnma@netcologne.de	Peter Johnson http://delphidabbler.com delphidabbler@gmail.com
Max Kleiner www.softwareschule.ch max @ kleiner.com	John Kuiper john_kuiper @ kpnmail.nl	Wagner R. Landgraf wagner @ tmssoftware.com
Vsevolod Leonov vsevolod.leonov@mail.ru		Andrea Magni www.andreamagni.eu andrea.magni @ gmail.com www.andreamagni.eu/wp
	Paul Nauta PLM Solution Architect CyberNautics paul.nauta@cybernautics.nl	Kim Madsen www.component4developers
Boian Mitov mitov @ mitov.com		Jeremy North jeremy.north @ gmail.com
Detlef Overbeek - Editor in Chief www.blaise-pascal.eu editor @ blaise-pascal.eu	Howard Page Clark hdpc @ talktalk.net	Heiko Rompel info@rompelsoft.de
Wim Van Ingen Schenau -Editor wisone @ xs4all.nl	Peter van der Sman sman @ prisman.nl	Rik Smit rik @ blaise-pascal.eu www.romplesoft.de
Bob Swart www.eBob42.com Bob @ eBob42.com	B.J. Rao contact@intricad.com	Daniele Teti www.danieleteti.it d.teti @ bittime.it
	Anton Vogelaar ajv @ vogelaar-electronics.com	Siegfried Zuhr siegfried @ zuhr.nl

Editor - in - chief

Detlef D. Overbeek, Netherlands Tel.: +31 (0)30 890.66.44 / Mobile: +31 (0)6 21.23.62.68
News and Press Releases email only to editor@blaise-pascal.eu

Editors

Peter Bijlsma, W. (Wim) van Ingen Schenau, Rik Smit

Correctors

Howard Page-Clark, Peter Bijlsma

Trademarks All trademarks used are acknowledged as the property of their respective owners.

Caveat Whilst we endeavour to ensure that what is published in the magazine is correct, we cannot accept responsibility for any errors or omissions. If you notice something which may be incorrect, please contact the Editor and we will publish a correction where relevant.

Subscriptions (2017 prices)

	Internat. excl. VAT	Internat. incl. VAT	Including Shipment
Printed Issue ±80 pages	€ 235	€ 266,50	€ 85,00
Electronic Download Issue 80 pages	€ 50	€ 60,50	—
Printed Issue Inside Holland(Netherlands) ±80 pages	—	€ 180	€ 30,00



Member and donator of WIKIPEDIA

Subscriptions can be taken out online at www.blaise-pascal.eu or by written order, or by sending an email to office@blaise-pascal.eu. Subscriptions can start at any date. All issues published in the calendar year of the subscription will be sent as well.

Subscriptions run 365 days. Subscriptions will not be prolonged without notice. Receipt of payment will be sent by email.

Subscriptions can be paid by sending the payment to:

ABN AMRO Bank Account no. 44 19 60 863 or by credit card or Paypal

Name: Pro Pascal Foundation-Foundation for Supporting the Pascal Programming Language (Stichting Ondersteuning Programmeertaal Pascal)

IBAN: NL82 ABNA 0441960863 BIC ABNANL2A VAT no.: 81 42 54 147 (Stichting Programmeertaal Pascal)

Subscription department

Edelstenenbaan 21 / 3402 XA IJsselstein, The Netherlands

Mobile: + 31 (0) 6 21.23.62.68 office@blaise-pascal.eu

Copyright notice

All material published in Blaise Pascal is copyright © SOPP Stichting Ondersteuning Programmeertaal Pascal unless otherwise noted and may not be copied, distributed or republished without written permission. Authors agree that code associated with their articles will be made available to subscribers after publication by placing it on the website of the PGG for download, and that articles and code will be placed on distributable data storage media. Use of program listings by subscribers for research and study purposes is allowed, but not for commercial purposes. Commercial use of program listings and code is prohibited without the written permission of the author.



From the editor

"Times they are a changin..."

an old song of Bob Dylan. So actually anything changes all the time: I was very much surprised these days when I was doing some research about the Quantum Internet that there will be soon some rigorous new developments to happen in two years: 2020!
Read the transcript (page 30) of this TEDx where Stephanie Wehner is explaining what the Quantum Internet means: A step for mankind as big as the creating of the internet in 1969, as big as Columbus discovering the Americas in 1492. And even bigger. Put it in to your Agenda...

I am always fascinated by these kind of amazing and even fantastical developments but we have done quite some developments of our own: (I mean we as a Pascal community).
We created Pas2 JS, the Webcore component group from TMS, the FNC component group - again from TMS and now we have a new phenomena:
the gaming suite...Why is that so special?

Because you now can make your own 2D and 3DGames. Because that means you can create your own 3D live software and because something very special is coming on that ever has been my greatest dream:
I want to make Pascal available to children in the most purest and simplest way:
let them play with Pascal. How?

Quite simple: this gaming engine makes it possible for a child to have a form where you can draw on, create your own house or bear or cat or car and... move it live around on the paper. Turn it around or make 3D of it. And...on the other form you immediately can see the code. You can see it changing live.
And backwards: if you change the code the tree will grow - or shrink! Live.

But that's not all: I would really like the kid to learn some of the directives that you need for pascal code writing: think of that apple falling from the tree - which Mr. Newton experimented with, then remember the steps: if - then else? What about an array? Pretty easy to draw... Imagine...

How fabulous would that be - explaining code to a child and let it try it by altering the statement through drawing it?
And vice versa...
learning all the essential bits and pieces through playing?

I will show it to you in Köln/Bonn next Saturday (22 September) with the help of **Michalis Kamburelis** who is the developer of all that beautiful stuff.

Imaging you your self could in code write a scenario where you can walk through a building and let your customer have a preview and tell you where the cables, the communications and what other things could go by walking through that building... and all that is done in ONE Language: Pascal!

I think we are quite well on the way to make Pascal the tool it should be: a great tool for everyone in whatever situation on whatever platform...

If this really interests you:
come to the Lazarus Professional Conference next weekend: 20/21/22 of September. The Saturday (22) is the best way for you to get value for your money: you will get the Library Credit Card USB Stick for free! If you come, send us your request for the voucher code and we will make sure you will get extra discount... office@blaisepascal.eu

Your editor,
Detlef Overbeek

TMS WEB Core v1.0 Brescia Available now!

WEB

Delphi RAD modern web client development starts here



web.tmssoftware.com

€ 295

Features

RADical Web

- Modern SPA web application model
- Pure HTML5/CSS3/Javascript based applications
- Standard component framework for common UI controls and access to browser features
- Debugging in Pascal code via the browser
- Backed by a solid & proven Delphi Pascal to Javascript compiler that was years in development

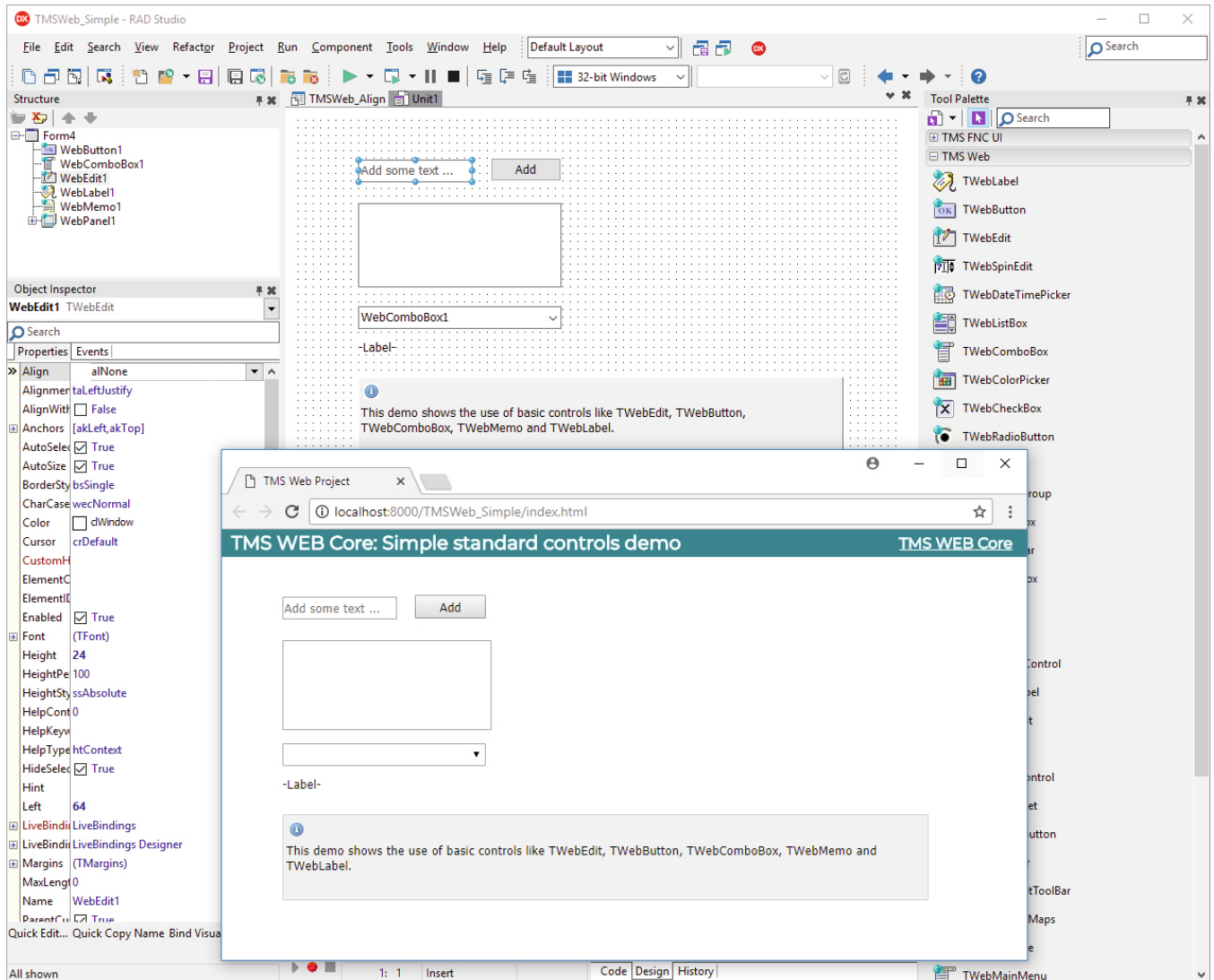
Reuse skills and components

- Component based RAD development integrated in the Delphi IDE
- A truly revolutionary & innovative TMS FNC component framework that is now also web enabled, allowing to create UI controls that can be used on VCL, FMX, LCL and WEB!
- Open to consume other existing Javascript frameworks & libraries
- Open to use HTML/CSS for design
- Open to use other jQuery controls or even other Javascript frameworks
- Offers Pascal class wrappers for jQuery controls from the jqWidgets library
- Easy interfacing to REST cloud services including to TMS XData for database

Easy Deployment

- Application consists of HTML & Javascript files only that can be easily deployed on any light or heavyweight webserver
- Use any existing load-balancing software and/or techniques for highest performance
- Small and convenient debug webserver is included for fast RAD development

<http://web.tmssoftware.com>



Demos Basics

- Simple
- Bootstrap
- Align
- Anchors
- Dataset
- GridPanel
- HTML template
- MessageDialogs
- Multiform
- PaintBox
- RichEditor
- Responsive Grid
- Table Control

FNC

- Chart
- Grid
- KanbanBoard
- Listbox
- Planner
- TVGuide
- Treeview
- Tableview
- NavigationPanel
- PageControl
- Grid Database Adapter

jQuery

- Overview
- Grid

Services

- Google Calendar
- myCloudData
- Simple

Web

- Embedding
- Geolocation

XData

- Web Client



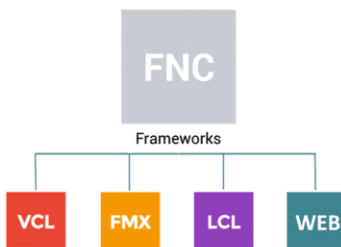
develop • fnc

Use one RAD UI control set everywhere.

Cross-platform & cross-framework

Framework Neutral Components mean:

- Only one UI control set
- Only one learning curve
- Only one license
- Freedom of choice to target Windows, macOS, Android, iOS, Linux and Web
- Freedom of choice to use VCL, FMX, LCL, TMS Web



Reuse single codebase

- Use 100x identical Delphi UI logic code in VCL, FMX, LCL or TMS Web apps
- One UI control set for VCL, FMX, LCL, TMS WEB frameworks
- Abstraction layer to write your code framework independent
- Component with 100% identical interface and look & feel in all frameworks and all platforms

Quality set of components

One solid code base for complex and feature-rich components:

- Grid
- Planner
- RichEditor
- Treeview
- Ribbon
- And more...



FNC TMS FNC Component Studio

TMS FNC Component Studio includes: TMS FNC UI Pack / TMS FNC Chart / TMS FNC Blox / TMS FNC Dashboard pack

FNC	Single Developer License
€ 295	

FNC	Small Team License
€ 495	

FNC	Site License
€ 995	

ALL TMS ALL-ACCESS

This product is also available in the following bundle(s): [TMS ALL-ACCESS](#)

develop • biz

A robust framework to build a REST API server



XData API Server

- Build a REST API server with Delphi and XData in minutes
- Publish all your database data automatically in REST endpoints with a few lines of code
- Easy JSON serialization and database retrieval
- Full query syntax for retrieving entities through the API



ORM and Database Development



Aurelius: State-of-the-art ORM framework for Delphi

- Abstract your database in objects: focus on business logic instead of persistence mechanism.

Data Modeler: Visual database design tool

- Reverse engineering of existing database
- Easy-to-use ER Diagrams

Quality Enterprise Software Tools

- Scripter: Add ultimate end-user customization to your application
- Workflow: Design and run business processes with task management
- Diagram Studio: Visual diagram/flowchart component
- Query: Make queries easy, flexible & powerful
- Echo: Automatic data replication



BIZ TMS BIZ

TMS Business Subscription includes: TMS Aurelius / TMS Data Modeler / TMS Sparkle / TMS RemoteDB / TMS XData / TMS Echo / TMS Scripter / TMS Diagram Studio / TMS Query Studio / TMS Workflow Studio

BIZ Single Developer License

€ 495

BIZ Small Team License

€ 745

BIZ Site License

€ 1695

ALL TMS ALL-ACCESS

This product is also available in the following bundle(s): [TMS ALL-ACCESS](#)

starter

expert



New developments and features in the Delphi programming language are very interesting. However, the purpose of programming languages is in it's applications: analysing statistical data, handling logistic problems or performing scientific calculations.

In this last catagory I present a Delphi project called statics which calculates the effect of loads that are hanging on a cable. This cable is non-elastic and it's own weight is neglectable. It is a nice application of high-school mathematics and physics. Look at the next picture for the program at work:

The description of this statics project comes in four parts:

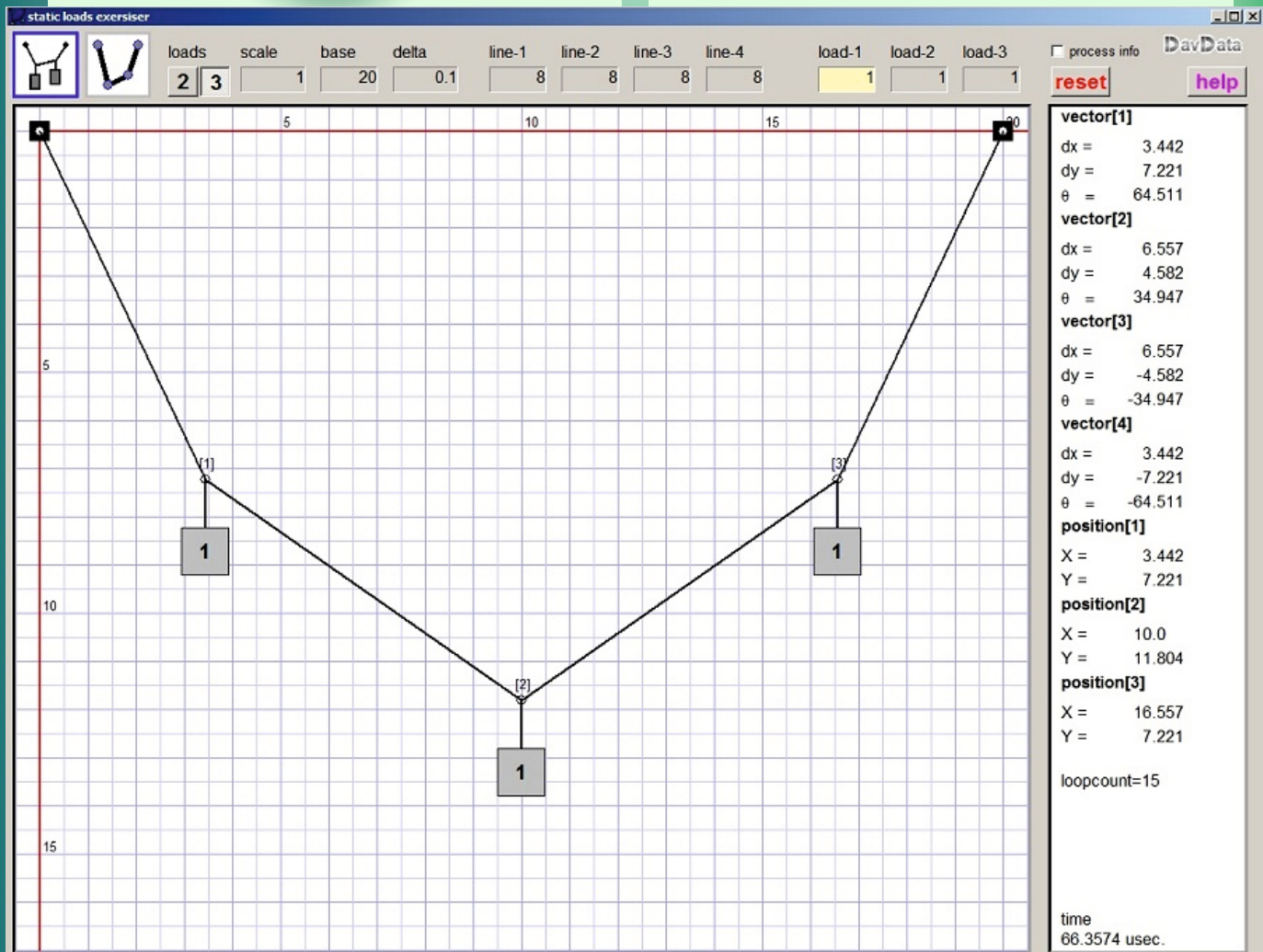
1. program description
2. theory, which includes some vector operations and trigonometry
3. Delphi implementation of calculations
4. menu buttons and program control

PROGRAM DESCRIPTION

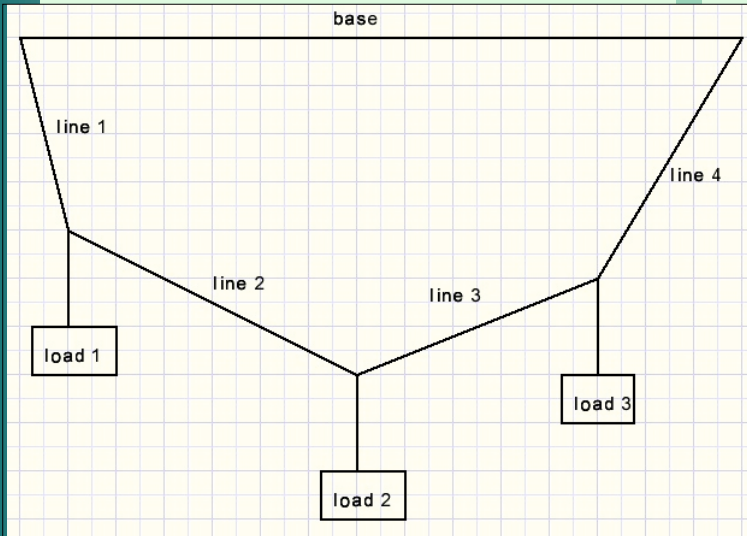
The program operates in either line mode or bar mode. In bar mode, the lines are bars of uniform weight, no loads are selectable.

The choice is between 4 or 3 lines with 3 or 2 loads, or 4 or 3 bars.

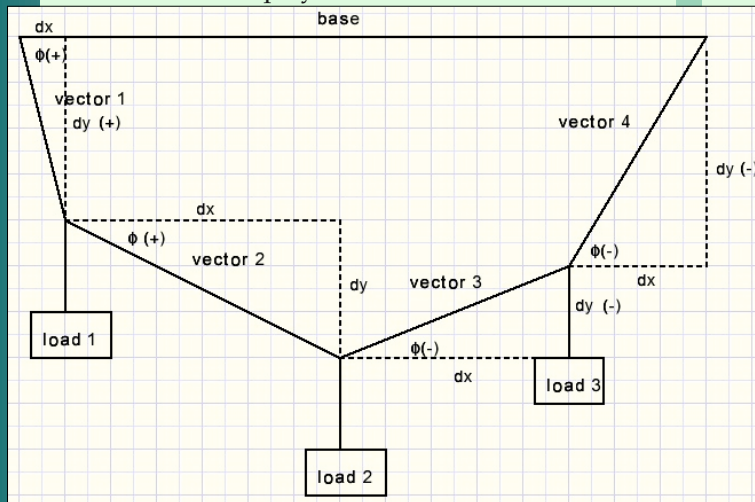
Above picture shows line mode with 3 loads.



Next picture shows the lines and loads:



If line lengths, bar lengths or loads are changed, a new balance is calculated and flowing information is displayed:



Note: just as on the Delphi canvas, the positive vertical (Y) direction is down. Please look at the controls at the top of the form, left to right we see:

line mode:	click on the left top line image to select
bar mode:	click on the bar image to select
3,4 lines/bars:	click on 3,4 button to select
Scale	display only, distance between two dark blue lines
Base	distance between cable attachment points
Delta	increment / decrement value for base, lines, bars, loads
Line 1..4	lengths of line 1..4
Load 1..3	weight of load 1..3
Reset	resets loads / lines or bars to default values.
Help	opens on-line help page.
Info box	adds additional information if calculations fail.

Making selections

To select the base, delta, lines or loads

- put mouse pointer over control and click or
- use cursor LEFT / RIGHT keys to switch to other control

Making changes

- use mousewheel to increment or decrement a selected value by delta or
- use cursor UP/DOWN keys.

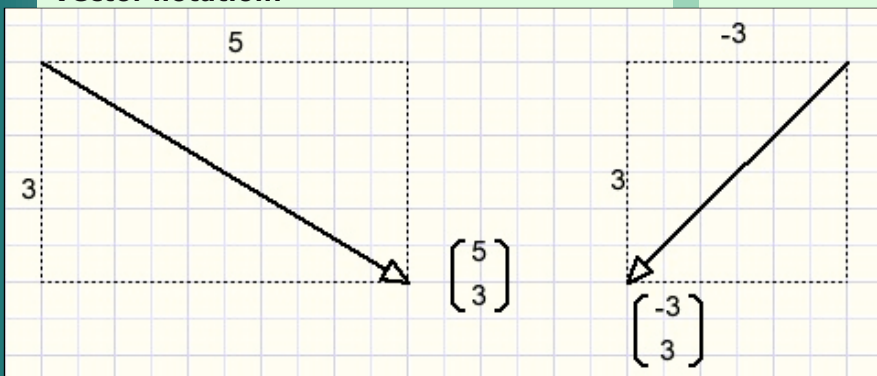
Theory

The variables in this project are the line lengths and the forces of the loads which are caused by gravity.

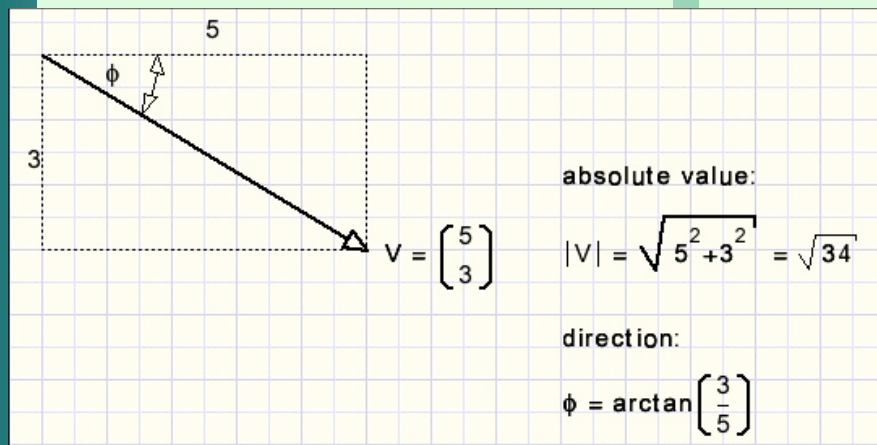
A line length is defined by a single number. A force however has a magnitude and also a direction in which the force is applied.

For that reason one number is not enough to define a force, we need two numbers which are called a vector.

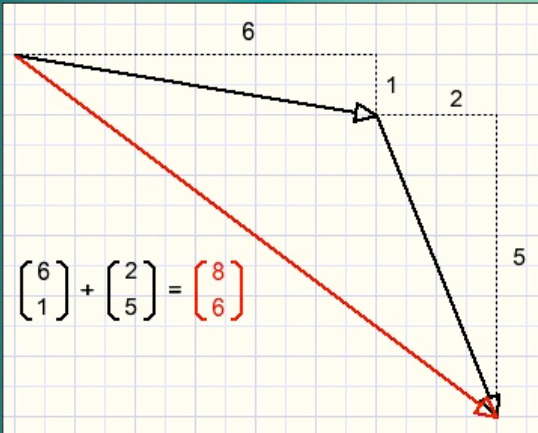
Vector notation:



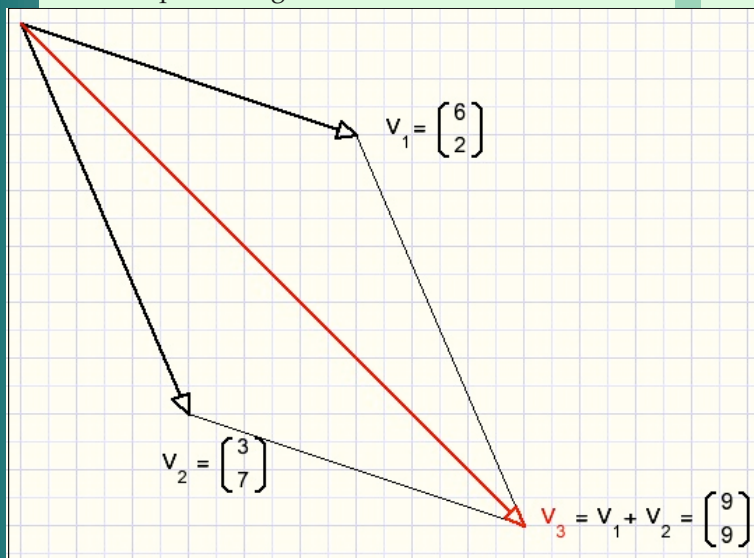
Absolute value and direction of a vector:



Vector addition (connect head to tail)



Addition of vectors working from a single point makes a parallelogram



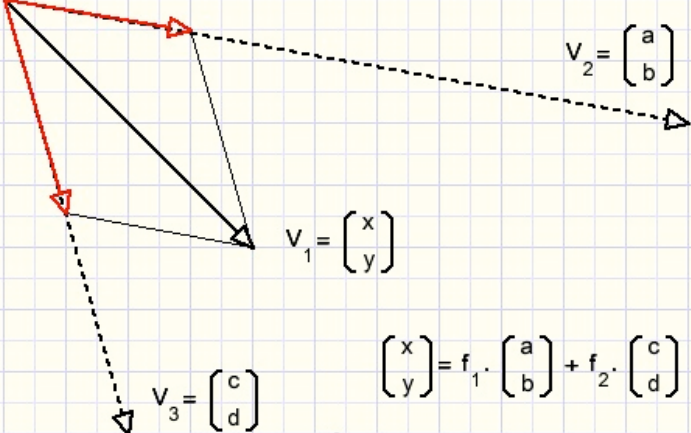
Multiplication of a vector by a single number



Vector splitting.

A vector may be written as the sum of two other vectors.

Say we split known vector V_1 in the sum of vectors V_2 and V_3 where vectors 2 and 3 only supply the direction:



$$V_2 = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$V_1 = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$V_3 = \begin{bmatrix} c \\ d \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = f_1 \cdot \begin{bmatrix} a \\ b \end{bmatrix} + f_2 \cdot \begin{bmatrix} c \\ d \end{bmatrix} \quad \dots(1)$$

$$\begin{cases} x = f_1 \cdot a + f_2 \cdot c & \dots * d \\ y = f_1 \cdot b + f_2 \cdot d & \dots * c \end{cases}$$

$$x \cdot d = f_1 \cdot a \cdot d + f_2 \cdot c \cdot d$$

$$y \cdot c = f_1 \cdot b \cdot c + f_2 \cdot d \cdot c$$

$$x \cdot d - y \cdot c = f_1 (a \cdot d - b \cdot c)$$

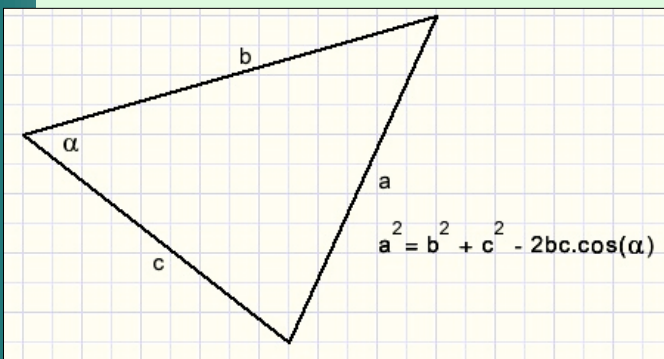
$$f_1 = \frac{x \cdot d - y \cdot c}{a \cdot d - b \cdot c} \quad \text{also : } f_2 = \frac{y \cdot a - x \cdot b}{a \cdot d - b \cdot c}$$

substitute f_1 , f_2 values in (1)

In the above picture black vector V is split in the two red vectors.

Of course there is the Pythagoras lemma, which is assumed to be common knowledge.

We need one more formula: the cosine rule to calculate an angle of a given triangle.



That's all, we do not need more theory to solve the cable load problems.

DATA FORMATS

Data definitions and calculations are all done in unit calc_unit.

```
type TVector = record
    dx : double;
    dy : double;
end;

var load : array[1..3] of double;
    line : array[1..4] of double;
    vector : array[1..4] of TVector;
    linecount : byte; // 3 or 4
    lineMode : Boolean; // false=bar mode
    base : double;
    scale : double; //distance value of 48 pixels
```

Note: the line[1..4] values are the absolute values of vectors[1..4]

CALCULATIONS

The program allows for the selection of 2 or 3 loads. In the case of 2 loads, the load positions are fixed if we know the angle of line 1 (the direction of vector 1).

In the case of 3 loads, we need the directions of vectors[1] and [4] to know all load positions. The direction of vector[1] is called a, the direction of vector[4] is b. (angles in radians)
See picture further below.

This article focusses on the calculations with 4 lines and 3 loads.

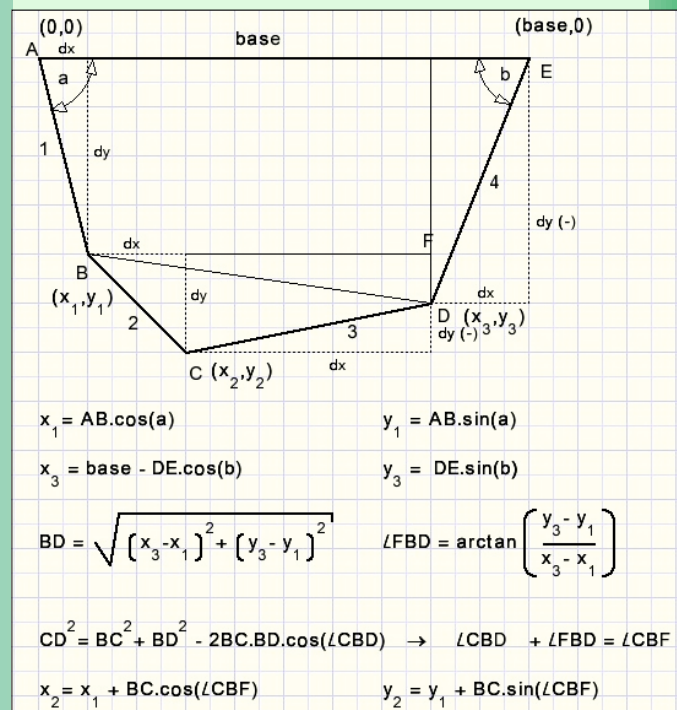
In the case of bar mode, the loads are calculated from the bar lengths and calculations are the same as in line mode.

load[1] is attached to B, load[2] to C, load[3] to D

AB = line[1] BC = line[2] CD = line[3]
DE = line[4]

Calculation of load positions B, C, D from values a and b :

vector[1].dx = x1 vector[1].dy = y1
vector[2].dx = x2-x1 vector[2].dy = y2-y1
vector[3].dx = x3-x2 vector[3].dy = y3-y2
vector[4].dx = base-x3 vector[4].dy = -y3



Start values for a and b

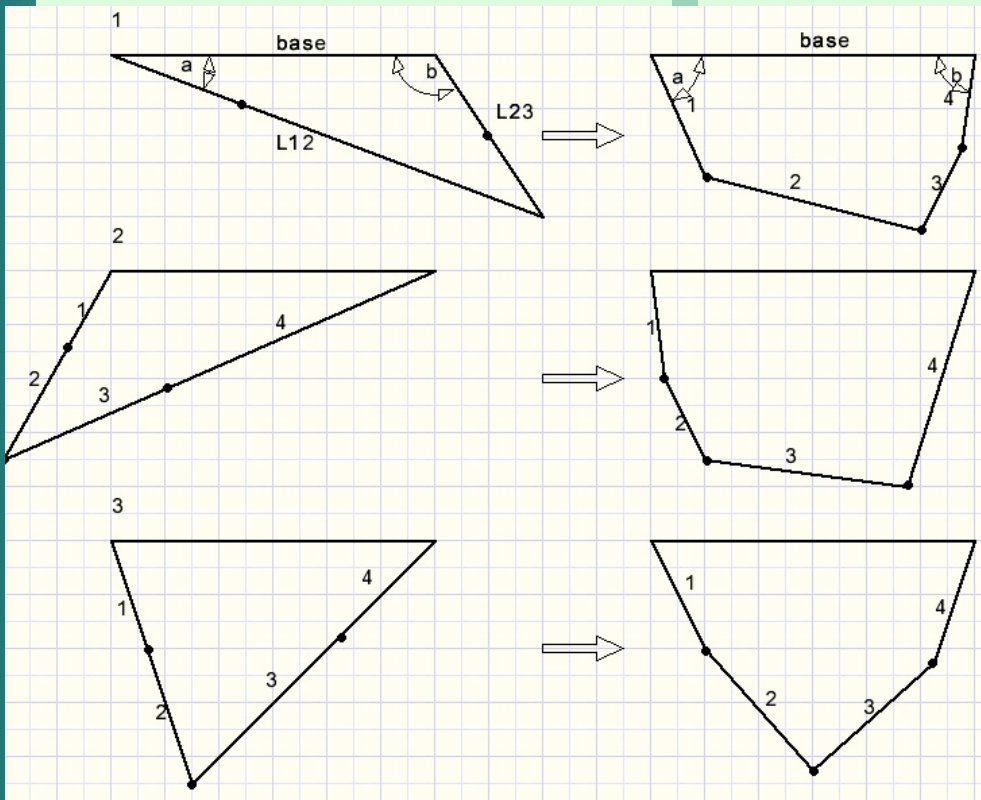
An equilibrium of forces is found by varying angles a and b.

Given line lengths 1..4, suitable angles for a and b must found for a start.

First $L_{12} = \text{line}[1] + \text{line}[2]$ and $L_{34} = \text{line}[3] + \text{line}[4]$ are calculated.

Imagine a triangle with edges base, L12 and L34. Depending on the angles a and b between base and L12, L34 the triangle is acute- or obtuse angled.

Three cases are considered:



The initial setup of angles a, b is done by

```
procedure setUpVectors4(var a,b : double);
//initialize vectors before calculation
var code : byte;
    L12,L34,x : double;
    w : Tvector;
begin
    dataOK := false;
    L12 := line[1] + line[2];
    L34 := line[3] + line[4];
    code := 0;
    if sqr(L12) + sqr(base) > sqr(L34) then code := 1;
    if sqr(L34) + sqr(base) > sqr(L12) then inc(code,2);
    case code of
        1 : begin           //a sharp
            b := pi2 - 0.02;
            w.dx := line[4]*cos(b) + line[3]*cos(b-0.02);
            w.dy := line[4]*sin(b) + line[3]*sin(b-0.02);
            x := sqrt(sqr(w.dy) + sqr(base - w.dx));
            if x < line[1] + line[2] then
                a := arctan(w.dy/(base-w.dx)) + getAngle(line[2],line[1],x);
            end;
        2 : begin           //b sharp
            a := pi2 - 0.02;
            w.dx := line[1]*cos(a) + line[2]*cos(a-0.02);
            w.dy := line[1]*sin(a) + line[2]*sin(a-0.02);
            x := sqrt(sqr(w.dy) + sqr(base - w.dx));
            if x < L34 then
                b := arctan(w.dy/(base-w.dx)) + getAngle(line[3],line[4],x);
            end;
        3 : begin           //a , b sharp
            a := getAngle(L34,L12,base)+1e-2;
            b := getAngle(L12,L34,base)+1e-2;
        end;
    end;
end;
if code > 0 then dataOK := makevectors4(a,b);
end;
```

Note: 0.02 radians = 1.15 degrees.

To find the angle of a given triangle, using the cosine rule:

function getAngle(a,b,c : double) : double;
returns the angle opposite edge a in radians. (pi radians = 180 degrees)

```
function getAngle(a,b,c : double) : double;
//cosine rule to find angle opposite edge a of triangle abc
//abc must be real triangle
var cs : double;
begin
    cs := (sqr(b) + sqr(c) - sqr(a))/(2*b*c);
    if (cs <> 0) and (cs <= 1) and (cs >= -1) then
        result := Vdir(sqrt(1-sqr(cs)),cs)
    else
        if cs = 0 then result := pi2
        else if cs > 1 then result := 0
        else result := pi;
    end;
end;
```

To find the direction of a vector:

```
function VDir(y,x : double) : double;
//vector direction in radians
//0..pi  0..-pi
begin
  if x = 0 then
    begin
      if y >= 0 then result := pi/2 else result := -pi/2;
    end
  else begin
    result := arctan(y/x);
    if x < 0 then
      if y >= 0 then result := pi + result else result := -pi + result;
    end;
  end;
end;
```

To calculate all vectors given angles a,b:

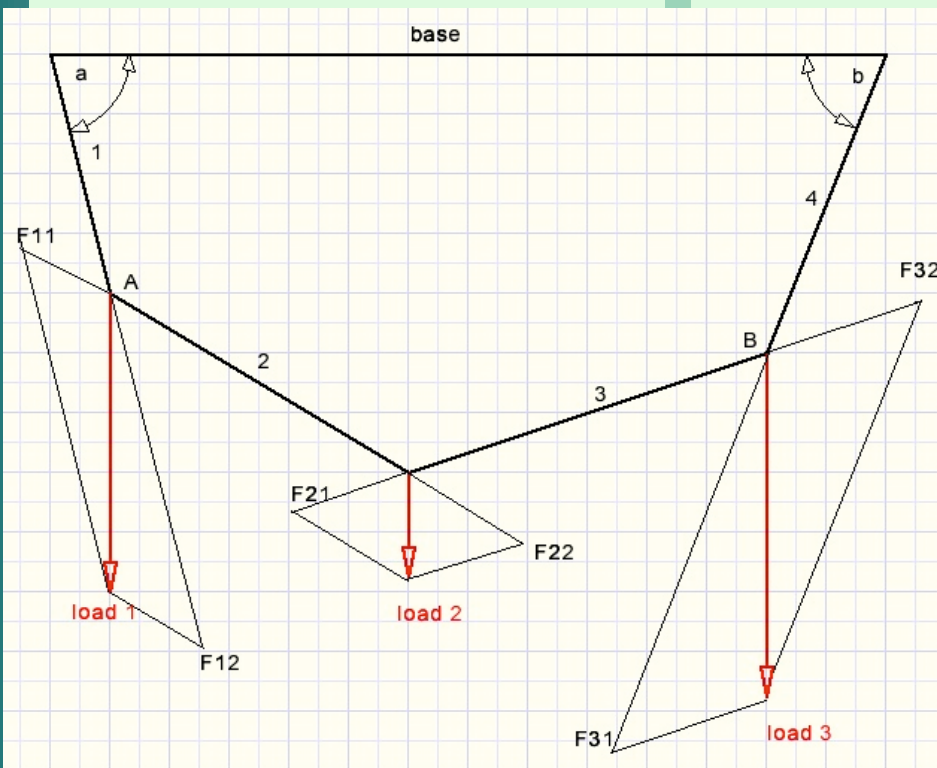
function makeVectors4(a,b : double) : boolean;

This function gives a false result if the construction was not possible.

Please refer to to source code.

Finding a balance

With the initial setup of angles a, b done we arrive at the core of this project :
adjusting a,b for an equilibrium.



Load 1 vector is split in vectors F12 and F11
 Load 2 vector is split in vectors F21 and F22
 Load 3 vector is split in vectors F31 and F32

F12 and F31 are cancelled by the tension in lines 1 and line 4.
 If F1 is the sum of forces working in point A:
 $F1 = \text{load}[1] + F11 + F22$

If F2 is the sum of forces working at point B:
 $F2 = \text{Load}[3] + F21 + F32$
 If F1 is negative (left) then angle a is increased, if positive a is decreased.
 If F2 is negative then angle b is decreased, if positive b is increased.

Any time a direction change (increase..decrease) takes place, the stepvalue is reduced by 50%.

This iterative process stops if both F1 and F2 forces are near zero.

The approximation process is performed by procedure calculate4;

Please refer to the source code.

Angles a and b have separate step values which increment or decrement them.

If the number of iterations reaches 1000, the process is aborted and a time-out message is reported.

Splitting a vector (writing as the sum of two other vectors) is at the core of the process:

```

procedure SplitVector(var v1,v2:TVector; v3:Tvector);
//call v1,v2: directions
//return v1 + v2 = v3
var d,f1,f2:double;
begin
  d:=v2.dx*v1.dy - v1.dx*v2.dy;
  if abs(d) < 1e-6 then
    begin
      f1 := 10000;
      f2 := -10000;
    end
  else
    begin
      f1 := (v2.dx*v3.dy - v3.dx*v2.dy)/d;
      f2 := (v3.dx*v1.dy - v1.dx*v3.dy)/d;
    end;
  v1.dx := f1*v1.dx;
  v1.dy := f1*v1.dy;
  v2.dx := f2*v2.dx;
  v2.dy := f2*v2.dy;
end;
    
```

PROGRAM CONTROL

This is done in unit1. User selections include

- switching between line mode and bar mode
- selecting 2 or 3 loads or 3 or 4 bars
- selecting the base
- selecting line lengths
- selecting loads (line mode only)

Line / bar mode selection is done by clicking on a Timage.

These images are painted at create time.

To show the selection, a 3 pixel blue line is painted around the Timage at the form1 canvas. The selections of line lengths and loads are shown in Tstatictext components.

The selected statictext background is colored yellow.

< and > cursor keys allow for the selection of a new statictext.

Cursor UP / DOWN key entries then increment or decrement the selected value.

Also the mousewheel may be used to change a selected value.

At create time, the statictext components are placed in array `labellist[...]`

This makes selection changes easier because now we use the index value to this list.

Line- and load component mouse events share the same methods.

The tag property of the line statictext is used to indicate the line number [1..4]

The tag property of the load statictext indicates the number of the load [1..3]

In several methods the code `activecontrol := nil` had to be included to avoid cursor key events being send to the reset- or help button or the info checkbox.

Painting of the raster and vectors is done in a bitmap called map which is copied to paintbox1 on form1 to become visible.

Information about vectors and their positions is painted directly into paintbox2 on form1.

To avoid unpleasant flickering when changing line lengths or loads, the painting in paintbox2 is done 300milliseconds after completion of the last update.

A TTimer component takes care.

Postscript.

Recently a large bridge collapsed in the Italian city of Genua.

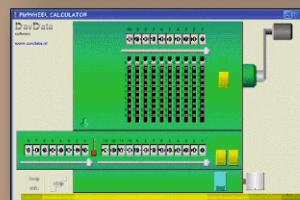
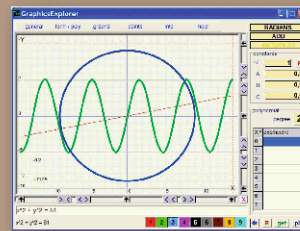
Forces working on a bridge are more complicated but similar to the forces in this project.

It is not possible to solve this type of problems by analytic means (solving systems of equations).

The only way is a numerical approach using successive approximation which involves very much calculations and requires a computer.

The Italian bridge was almost 60 years old, calculations were done by hand at that time.

DAVID DIRKSE



```
procedure ;
var
begin
  for i := 1 to 9
  do
    begin
```

BLAISE PASCAL MAGAZINE 

www.blaisepascal.eu

COMPUTER MATH & GAMES IN PASCAL

SPECIAL OFFER COMBINED WITH THE LIBSTICK

<https://www.blaisepascalmagazine.eu/product/renewal-special-offer/>

SPECIAL OFFER COMBINED WITH THE LIBSTICK AND FREESUBSCRIPTION FOR ONE YEAR

<https://www.blaisepascalmagazine.eu/product/bundle-computer-graphics-math-games-pascal-libstick-download-subscription/>

CONTENT OF THE BOOK

<https://www.blaisepascalmagazine.eu/product/books/>



LAZARUS 2.0 ON THE EVENT:

Subjects on the event (Mattias Gärtner)

LCL Interfaces Changes / LCL Changes

TScrollingWinControl (TForm, TScrollBar, TFrame)

Added flags to exclude some graphics format to create smaller applications:

TCustomImageList

TImageList

TSpeedButton, TBitBtn

TWinControl.DoubleBuffered, .ParentDoubleBuffered

and TApplication.DoubleBuffered

IDE Changes:

Editor

Debugger

IDE Interfaces Changes / Components

TOpenGLControl/ TACHartChanges affecting compatibility: LazUtils /

LCL incompatibilities

TToolBar children ignore Align

TCustomComboBox.ReadOnly was deprecated

Predefined clipboard format pcfDelphiBitmap was removed

TEdit.Action visibility lowered to public

TControl.ScaleFontsPPI, .DoScaleFontPPI parameter change

MouseEntered deprecated/missing

TCustomImageList.Add method

TCustomTreeView.OnChanging event: Node parameter

No LCL Application exception dump

No default LazLogger

Screenshot for LCLExceptionStackTrace and LazLogger

Additions and OverridesComponents incompatibilities

LazControls: TSpinEditEx no longer inherits from TCustomFloatSpinEditEx

TACHart: Reticule-related properties deprecated

TACHart events On[After|Before]Draw[Background|BackwallWall] deprecated

TACHart:

The TCubicSplineOption csoDrawFewPoints is removed.

IDE incompatibilities



starter expert



Introduction

There have been some questions about how to build a server, with authorization and login management, where the users and their roles are defined in a database.

This blog post explains one way of doing that using the TkbmMwAuthorizationManager. Please refer to the previous post (REST easy with kbmMW #4 – Access management) for additional general information. First we should have some server that needs login support. For this sample, I have chosen the FishFact REST server, which was built in the blog REST easy with kbmMW #12 – Fishfact demo using HTTP.sys transport. See Issue nr 71/page 66

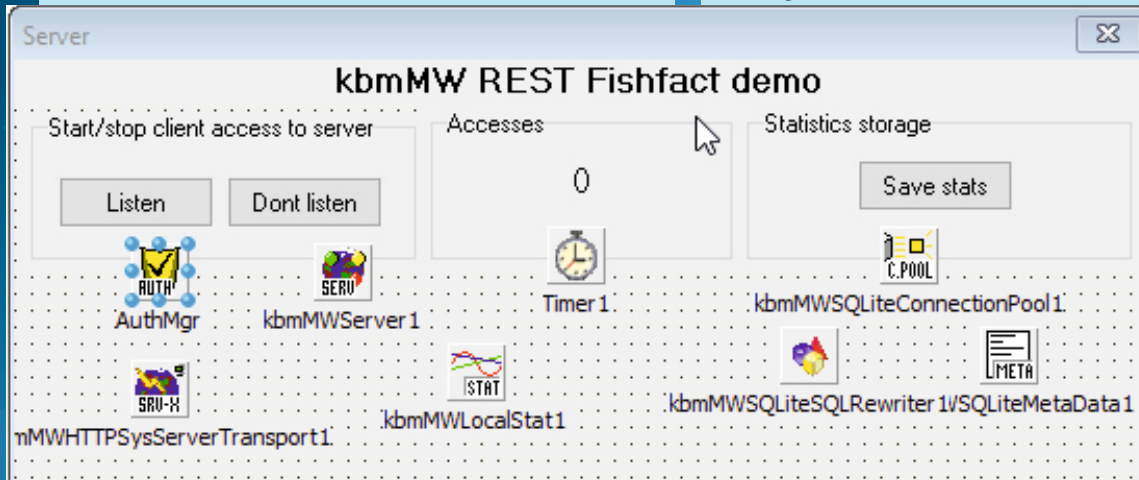


spiral galaxy M51a

Then we need to determine how to store and access user information from the database. Since this sample already uses the **ORM** (Object-Relational Mapping) to access the database, it makes sense to continue to do so for the user management. Let's add a class describing a user:

ADDING LOGIN SECURITY

Based on that server we add a **TjkbmMWAuthorizationManager** to the main form (Unit1).



```
[kbmMW_Table('name:user')]
TUser = class
private
    FID:kbmMWNullable<string>;
    FName:kbmMWNullable<string>;
    FPassword:kbmMWNullable<string>;
    FRole:kbmMWNullable<string>;
public
    [kbmMW_Field('name:"id"', primary:true, generator:shortGuid',ftString,38)]
    property ID:kbmMWNullable<string> read FID write FID;

    [kbmMW_Field('name:"name"',ftString,50)]
    [kbmMW_NotNull]
    property Name:kbmMWNullable<string> read FName write FName;

    // A secure system should never store plain text passwords, but only SHA256 hashed ones.
    // In that case make room for 64 characters.
    [kbmMW_Field('name:"password"',ftString,50)]
    property Password:kbmMWNullable<string> read FPassword write FPassword;

    [kbmMW_Field('name:"role"',ftString,30)]
    property Role:kbmMWNullable<string> read FRole write FRole;
end;
```



Notice the warning about the password. In this sample we store the unhashed plaintext password in the database. That is a NO NO in a production system. Instead one should store a hashed and salted version of the password... I'll explain later how to modify the code to do that.

For now we accept that the password is unhashed in the database.

In the already existing **Form.OnCreate** event handler, we should let the **ORM** ensure that the user table is made available. In addition we also should define the roles that are accepted by this server.

In our sample, there are only two types of users... the anonymous ones and the ones that are logged in with administrator rights. Most of the functionality is made available for anonymous users to use, except one **REST** call, which require the administrative role. But first things first:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
    FORM:=TkbmMWORM.Create;
    FORM.OpenDatabase(kbmMWSQLiteConnectionPool1);
    FORM.CreateOrUpgradeTable(TUser);

    // Add the one single role this application server knows about except anonymous.
    AuthMgr.AddRole('AdminRole');

    kbmMWServer1.AutoRegisterServices;
end;
```

The interesting parts here is the **CreateOrUpgradeTable** call, which ensures there is a table named user in the database, and the definition of a role called **AdminRole**.

In **Unit1** we should also remember to register the TUser class so **kbmMW** is aware about its existence. One place to do that is the initialization section of the form unit.

```
...
initialization
    TkbmMWRTTI.EnableRTTI([TUser]);
    kbmMWRegisterKnownClasses([TUser]);

end.
```

Now we must link the knowledge about the user table with the login process of the authorization manager. The crucial point is that the authorization manager is the supreme authority in relation to logins, and as such must know about the actors that are allowed to login. Thus the actors needs to be defined. It can either be done at startup time of the application server, where a complete list of known users are defined as actors towards the authorization manager, or alternatively it can be done on the fly on a need to know bases, which is what I have chosen to show here.

We use the **OnLogin** event of the authorization manager:

```

procedure TfrmMain.AuthMgrLogin(Sender: TObject; const AActorName,
  ARoleName: string; var APassPhrase: string;
  var AActor: TkmmMWAuthorizationActor; var ARole: TkmmMWAuthorizationRole;
  var AMessage: string);
var
  user:TUser;
begin
  // Lookup user with given name and password.
  user:=ORM.Query<TUser>(['Name', 'Password'],[AActorName,APassPhrase]);
  if user<>nil then
    try
      // Check if users role is defined. If not complain.
      ARole:=AuthMgr.Roles.Get(user.Role.Value);
      if ARole=nil then
        AMessage:='Role not supported'
      else
        begin
          // Check if actor exists, use it, else create one.
          AActor:=AuthMgr.GetActor(AActorName);
          if AActor=nil then
            AActor:=AuthMgr.AddActor(AActorName,APassPhrase,ARoleName);
          AMessage:='User found and is allowed login';
        end;
      finally
        user.Free;
      end
    else
      AMessage:='User not found';
    end;
  end;

```

Basically it use the **ORM** to lookup a user with the given name and password in the database. If one is found, it checks to see if the role that has been defined in the database on the user, exists.

If it does, then it attempts to figure out if the user has already been defined as an actor in the authorization manager. If not then one is defined and all is well.

What if the database is changed... For example if a user changes password? In such case you should not only update the password in the database but also update it in the actor representation in memory.

You can do something like this:

```

procedure TUnit1.UpdateUserPassword(const AUserName,
  ANewPassword:string);
var
  user:TUser;
begin
  AuthMgr.ChangeActorPassword(AUserName,ANewPassord);
  user:=ORM.Query<TUser>(['Name'],[AUserName]);
  if user<>nil then
    try
      user.Password:=ANewPassword;
      ORM.Update(user);
    finally
      user.Free;
    end;
  end;
end;

```



And if the user is to be deleted:

```
procedure TUnit1.RemoveUser(const AUserName:string);
begin
  AuthMgr.DeleteActor(AUserName);
  ORM.Delete<TUser>(['Name'],[AUserName]);
end;
```

Finally we should define exactly what should be protected by login.

For that we open **Unit2.pas** which contains the **REST** service, and choose one or more of the methods to protect. In this case **GetSpecieByCategory** will now require login as an administrative role, for it to be used.

```
[kbmMW_Rest('method:get, path:"specieByCategory/{category}"')]
[kbmMW_Auth('role:[AdminRole], grant:true')]
function GetSpecieByCategory([kbmMW_Rest('value:"{category}"')]
  const ACategory:string):TBiolifeNoImage;
```

Also remember to add **kbmMWSecurity** to the interface uses clause of the unit.
kbmMWSecurity.pas contains the definition of the **kbmMW_Auth** attribute.

Now we are done. Make sure to add a user with a password and a role of **AdminRole** to the user table, run the application server and try out the various **REST** calls.

The moment you try to this call:
http://localhost:1111/biolife/specieByCategory/Butterflyfish

You will be requested for a login by the browser. If you provide the user name and password matching a user in the database having the role **AdminRole**, you will be shown the result of the request. Otherwise you will only have access to all other **REST** calls which have no **kbmMW_Auth** attribute and as such are allowed to be called anonymously.

HASHING PASSWORDS

Remember that I mentioned storing (and transferring) plaintext passwords is a no no in production environments?

Hence we should encrypt the password before storage and transfer. However encryption typically means its possible to reverse the encryption, provided the encryption key can be guessed or hacked, which would reveal the password in plain text again. Since users may sometimes reuse the same password on multiple servers, we should make it as hard as possible for potential hackers to get back to the plaintext version of the password.

In a **REST** setup, its usually a web browser that decides how usernames and passwords are sent. The typical way is actually to leave all encryption to a **SSL** and user certificates and such to ensure that not only transmitted login data is scrambled, but also all other traffic between the browser and the server. Check the **REST easy with kbmMW #3 – SSL** blog post for information about one way to secure your **REST** application server with **SSL** and certificates.



And that is all fine and good, but we also have the storage part. We really shouldn't store the password in plaintext.

So we will use one way encryption... also known as hashing. It basically calculates a (complex) sum of the original password. Since its a "sum", its usually impossible to reverse the calculation back to the original password, provided a good secure hashing algorithm is used. Fortunately **kbmMW** provides native support for several secure hashing algorithms. One of the most used ones, that is generally considered secure, is called **SHA256**.

Now when we receive a password in the **OnLogin** event, we need to hash it before we do anything with it. It is super simple to do so.

Include **kbmMWHashSHA256** in your uses clause.

```
var
  hashed:string;
begin
  hashed:=TkbmMWHashSHA256.HashAsString(APassPhrase,'somesaltvalue');
...
end;
```

somesaltvalue is some "secret" value you have in your application and that is unique for your application. It can be anything, but prefer a length string of scrambled random characters.

The idea behind a "salt" is that it makes it extremely difficult to attempt to bruteforce guessing the correlation between a plaintext attempt and a calculated **SHA256** value. If you simply use the password by itself, then attackers has a much easier time attempting to guess the password, simply because they can try out all combinations of characters and match the hashed result with the sniffed hash value that you have hashed.

Adding a salt, ensures that the attacker will have no chance in brute force attacking by trying out all combinations, because regardless of what the attacker attempts to find, it will never be the same as the value you have stored in the database due to the secret salt.

Now we have a hashed string, and that one can be stored in a database, and similarly every time we need to figure out if person has typed in the correct password, we first need to hash it server side (with the correct salt) and then attempt to look the hashed password (and username) up in the database.

PROLOGUE

There are many more features in the authorization manager, which I have not explained here, but visit our site at <http://www.components4developers.com>, and look for the kbmMW documentations section for whitepapers.

If you like this, please share the word about **kbmMW** wherever you can and feel free to link, like, share and copy the posts of this blog to where you find they could be useful.

Oh... and whats about that featured image? It's an image of the spiral galaxy **M51a**, also known as the whirlpool galaxy. Whirlpool is also the name chosen for one of the stronger hashing algorithms, for which there has still not been found any significant weaknesses or attack vectors.nd they could be useful.



- KBMMW SMART CLIENT ON NEXTGEN (ANDROID) – SCOPE PROBLEMS

I've just discovered an issue when using the **kbmMW** smart client in a slightly more advanced way, on an Android device.

The **kbmMW** smart client use a special type of custom variant supported by Delphi, descending from **TInvokeableVariantType**, that allow writing calls to procedures/methods/functions that actually do not exist in the project.

It is a nice way to allow "embedding" script like functionality directly using an almost normal Delphi syntax, or in **kbmMW's** case to allow calling server side methods without having to write stub/skeleton code.

The following code is a call from a **kbmMW** client to a service (someservice) in a **kbmMW** server. The call accepts 4 arguments, an ID (string), an integer (100) and two generic **TObjectLists** (x.List1 and x.List2).

One of the caveats of using the **TInvokableVariantType**, is that one can only use arguments that can be stored within a variant. So all the regular types, integer, int64, string, float etc. are perfectly fine to use, but objects can't be passed on automatically, without doing some magic.

The magic in this case, is "casting" the object instance to a custom variant type which also supports controlling ownership of the object. Hence the **Use.AsVariant(...)** syntax which returns a special type of variant.

All this works perfectly fine on all platforms... as long as there are only one **Use.AsVariant** in the argument list.

On Android (and I assume **NextGen** in general), the variants are being deallocated before the call is actually being made, hence invalidating the contents of the record structures holding the relevant data, resulting in an "Invalid variant type" exception being thrown when attempting to run it on **NextGen**.

Whats the solution? Well there are multiple... one of them would be to combine List1 and List2 in a 3rd object which is sent... but that would require the server also being updated to support receiving a combined object.

Fortunately there is an easy solution:

```
var
  cli:TkbmMWSmartClient;
  v1,v2:variant
begin
  cli:=TkbmMWSmartRemoteClientFactory.GetClient(
    transport,'someservice');

  v1:=Use.AsVariant(x.List1,false);
  v2:=Use.AsVariant(x.List2,false);
  cli.Service.SomeMethod(x.ID,100,v1,v2);
...
end;
```

Define the variants as local variables, to prevent the compiler to prematurely deallocate the variant contents. Now everything works fine again, also on **NextGen**.

```
var
  cli:TkbmMWSmartClient;
begin
  cli:=TkbmMWSmartRemoteClientFactory.GetClient(transport,'someservice');

  cli.Service.SomeMethod(x.ID,100,Use.AsVariant(x.List1,false),Use.AsVariant(x.List2,false));
...
end;
```



starter

expert



INTRO DUNCTION

In a previous blog, I wrote about random numbers and password generators. One of the things that were shown, was that computer generated random numbers are not really random, but are calculated according from a base line (called a seed).

Different algorithms produce different quality of randomness,. Delphi's 32 bit built in random generator was shown to be very weak, and absolutely not recommended for anything security related.

The blog post touched other algorithms that generally are considered "secure" for most purposes. However the values are still calculated and are only depending on the seed value and hence is reproducible if you know the original seed value and the number of times the algorithm has been called.

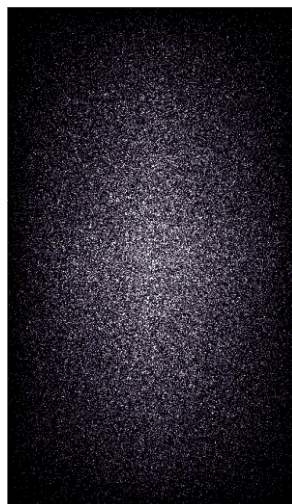
Next release of **kbmMW** solves that problem with support for **TRNG** (True Random Noise Generators). They are hardware based and they produce random noise based on random input from the real world.

within the Usually Delphi's built in 32 bit random generator is sufficient for most tasks, like generating random numbers for some tests, or a game or something similar. However the random generator is, if one focus on security, not strong enough to be used for cryptographic uses, like password generation.

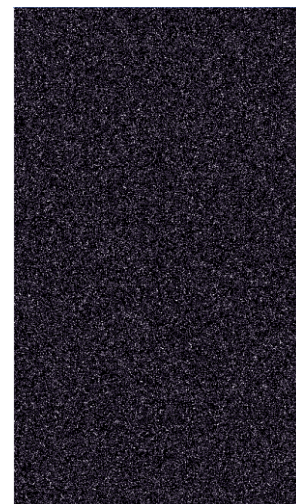
RANDOM NUMBERS

Using an updated version of the Random/password generator demo, included with kbmMW, I have generated 32 and 64 bit random values based on the hardware generator. The result is shown next to the previously shown randomness graphs.

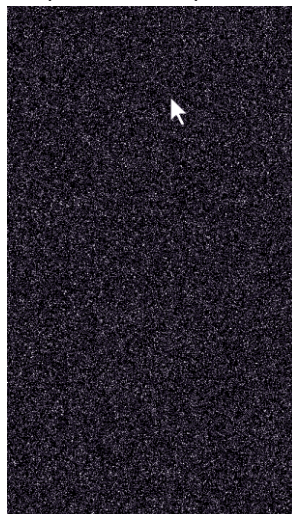
DELPHI'S 32 BIT BUILT IN RANDOM GENERATOR WAS SHOWN TO BE VERY WEAK, AND ABSOLUTELY NOT RECOMMENDED FOR ANYTHING SECURITY RELATED.



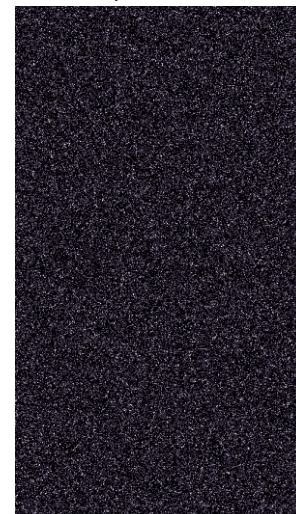
The 32 bit standard Delphi random plot



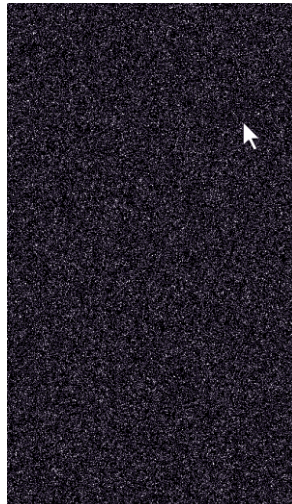
The 32 bit PCG random plot



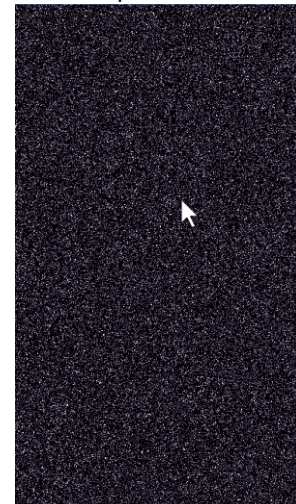
The 32 bit Mersenne Twister random plot



The 64 bit split mix random plot

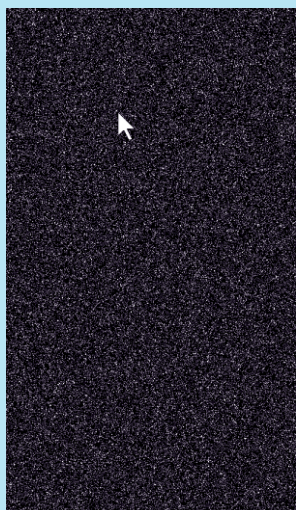


The 64 bit Xoroshiro 128 + random plot

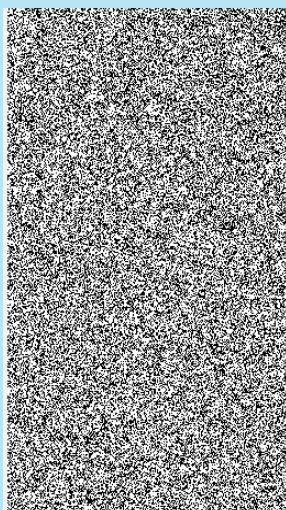


The 64 bit Xoroshiro 1024 random plot

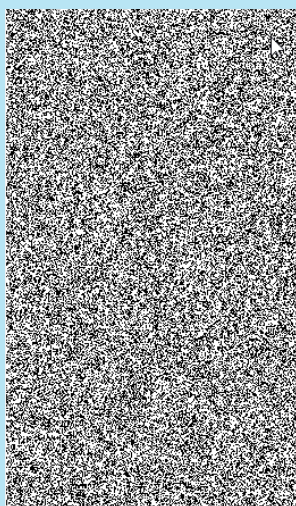




The 64 bit Mersenne Twister random plot



32 bit HW TRNG random plot



64 bit HW TRNG random plot

You can read more about the particular HW based random noise generator that kbmMW default supports here:

<https://13-37.org/en/infinite-noise-trng/>



DESCRIPTION

The Infinite Noise TRNG combines the best of two worlds. Modular entropy multiplication is used in the hardware to create provable random data. It's cryptographic strength is based on the SHA-3(Keccak) hashing function, implemented in software.

The hardware continuously reseeds that Keccak sponge
https://keccak.team/sponge_duplex.html
 to make it even more secure.
 So even if an attacker could get a snapshot of the Keccak state, it's worthless within a single cycle.

Raw data from modular entropy multiplication has certain properties (actually non-randomness) which are extremely useful:

The device is not rapidly changing the sort of numbers it puts out, so history can be used as a guide.

There is no special state stored in the modular entropy multiplier that could cause data to be different each clock cycle, other than on even/odd cycles.

Bits further away are less correlated.

This allows health monitoring on the raw data. Only when it's in the expected range, it is passed to the Keccak sponge.

Of course this also means you should not use the raw output for any cryptographic operations. But you have the freedom of accessing it directly to verify correct operation of the hardware or to apply different whitening functions, but right now SHA-3 is the best option!

WHATS THE DRAWBACK?

Well, as it reacts to real world data, it is somewhat slower in generating massive amounts of random values. kbmMW attempts to circumvent this problem by ensuring random data is generated even when you have not asked for it. But asking for loads of random values (like these plots) do take longer time than using any of the algorithmic random versions.

It is very much possible to combine the best of the fast algorithmic generators with this HW based one, for example by seeding the algorithmic ones regularly with values produced by the HW generator.

Then you will have something approximating true randomness with the high speed provided by the algorithms.



TEDx Vienna
x = independently organized TED event



INTRODUCTION:

The goal of a Quantum Internet is to connect quantum processors using long distance Quantum Communication. The internet has had a revolutionary impact on our world. The long-term vision of this talk is to build a matching Quantum Internet that will operate in parallel to the internet we have today.

This Quantum Internet will enable long-range quantum communication in order to achieve unparalleled capabilities that are provably impossible using only classical means.

Stephanie starts by exploring what a quantum internet is good for, and gives an intuition why quantum communication is so powerful. She proceeds from the state of the art today, towards stages for a full blown quantum internet.

As an example, she discusses the efforts of the EU quantum internet alliance including the planned demonstration network connecting four Dutch cities in 2020.

Stephanie Wehner is an Antoni van Leeuwenhoek Professor at QuTech, Delft University of Technology, where she leads the Quantum Internet efforts. Her passion is the theory of quantum information in all its facets, and she has written numerous scientific articles in both physics and computer science. In a former life, she worked as a professional hacker in industry. This talk was given at a TEDx event using the TED conference format but independently organized by a local community.

THE TALK

[HTTPS://WWW.YOUTUBE.COM/WATCH?V=XZPI2906DAC](https://www.youtube.com/watch?v=XZPI2906DAC)

I want to ask you a question what happened on the 29th of October 1969.

- Landing on the moon?

Close - but on this day something equally enormous happened, namely the first message was sent over the Internet.

So let me explain just how difficult it actually was to send that first message

The internet was super tiny and at half-past 10:00 in the evening researchers had gathered down in **Los Angeles** and at **Stanford** up in **Menlo Park** to send the first test message and they had agreed

to send the message: **LOG IN.**

Like looking into a remote computer and because this was challenging they called each other on the phone to make sure the message arrived.

So there were these guys down in **Los Angeles** at the computer and they type the first letter **L** and he asked on the phone did you see the L?

And an excited message came back: **yes, yes we see the L** and we sent the first letter over the internet.

They typed the second letter **O** and asked again did you see the **O** and they said yes: we also see the **O** and they typed the letter **G** and the system crashed.

So the Internet is a little bit larger now and we use it to send vast quantities of data.

I've recently moved into a new apartment and - just like in many places in the world - I sign up for internet like you sign up for power or water.

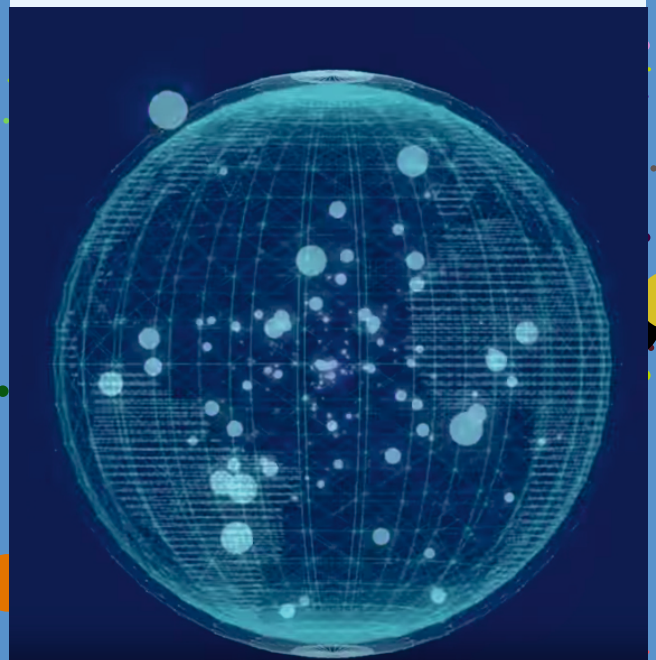
The Internet is everywhere and it certainly changed the way that we live today.

So I'm working on a quantum internet because I believe that it will similarly revolutionize the world.

Of course you're asking now:

WHAT IS THIS QUANTUM INTERNET?

On the surface it looks a lot like a classical Internet. There's computers and we're going to send data from one computer to another - only on the Quantum Internet this is a Quantum computer and I'm sending **Quantum Data**.



Quantum bits or qubits from one place to another. So these qubits are pretty special for example. We cannot copy a qubit like a classical bit that can be either 0 or 1.

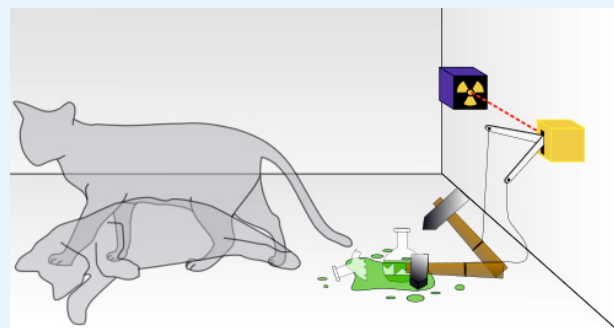
A **qubit** can be both 0 and 1 at the same time. You might imagine that instead of sending pictures of a cat that is either dead or alive we are going to send a Schrodinger's cat that is both dead and alive or 0 and 1 at the same time.



WIKIPEDIA

Schrödinger's cat: a cat, a flask of poison, and a radioactive source are placed in a sealed box. If an internal monitor (e.g. Geiger counter) detects radioactivity (i.e. a single atom

decaying), the flask is shattered, releasing the poison, which kills the cat. The Copenhagen interpretation of quantum mechanics implies that after a while, the cat is simultaneously alive and dead.



*Schrödinger's cat is a thought experiment, sometimes described as a paradox, devised by Austrian physicist **Erwin Schrödinger** in 1935. It illustrates what he saw as the problem of the Copenhagen interpretation of quantum mechanics applied to everyday objects.*

*The scenario presents a cat that may be simultaneously both alive and dead, a state known as a quantum superposition, as a result of being linked to a random subatomic event that may or may not occur. The thought experiment is also often featured in theoretical discussions of the interpretations of quantum mechanics. Schrödinger coined the term **Verschränkung (entanglement)** in the course of developing the thought experiment.)*

So these qubits are pretty cool but it brings us to the question why do we want to send them around? A quantum Internet has many applications that are impossible on the Internet of today.

It can make many things more efficient and in fact to use the Quantum Internet you also don't need to have a large quantum computer at home. Probably the most famous application of a Quantum Internet is to use it for secure communication.

SECURE COMMUNICATION

So secure communication means that I am going to send data to you like my credit card information.

Or maybe State Secrets, and I want that no one is able to listen into this communication.

A quantum internet allows us to use quantum key distribution that enables secure communication, whose security relies only on the laws of quantum mechanics in particular.

It's secure even if some eavesdropper was trying to snoop into our communication - has a quantum computer - and it remains secure even if this eavesdropper buys a quantum computer tomorrow.

So that's pretty nice but the Quantum Internet has many other, possibly less well-known applications, which personally I find probably even more exciting using a Quantum Internet.

We can use a quantum computer in the cloud. So quantum computers promise to solve many problems much faster than a classical computer. In particular we think that they're extremely good at simulating problems in quantum chemistry.

So you might imagine that in the future - if you want to explore a new material or a new medicine, you don't first go into the lab with some hugely long time consuming experiment. You're first going to run the quick simulation on your quantum computer to see whether that makes sense at all.

Now quantum computers are pretty expensive I sort of expected. I will actually be dead by the time that we all have a quantum computer at home or in our office but of course we want to use this computing power, so the first quantum computers are likely going to be somewhere in the cloud and I'm going to pay, say to use that for an hour.

Now maybe I want to use this quantum computer to perform a simulation on some proprietary material or possibly even my own DNA. But I don't want to give my DNA to this quantum computer.

It turns out that using a Quantum Internet we can use this remote quantum computer in the cloud securely in the sense that it has no idea what we're going to use it for.

Quantum Internet can also do all kinds of other cool things like it's good at synchronizing clocks much more accurately than we can do classically. It can be used to enhance password identification to some server far away and curiously we can even use it to play better at an online game. So what is it about these qubits that is so special that it enables all these applications to qubits can be entangled using a Quantum Internet.

Now entanglement also still sounds pretty mysterious - but there's actually only two features of quantum entanglement that lie at the bottom of all of these applications. If you understand these two features you will have a pretty good intuition of what a quantum Internet is good for.

WHAT ARE THESE TWO FEATURES?

The first one is called "**maximum coordination**" let's say I've used the Quantum Internet to entangle a qubit here in Vienna with some cubed far away somewhere down in Sydney and I'm going to perform a measurement on my qubits and let's imagine a friend down in Sydney performs exactly the same measurement. Now if you make the same measurement we will get the same measurement outcome instantaneously. You can think of a measurement as asking a question to a qubit.

I can ask qubits are you pointing left or are you pointing right? And if I ask the question to Mike you are here in Vienna and my friend asks the same question down in Sydney, then if I see left he will see left and if I see right he was seeing right.

And this happens instantaneously even though the answer is not determined ahead of time. The cool thing is that it actually works for any

If I had asked are you red or green, we would have seen exactly the same. We get always the same answer - so entanglement is maximally coordinated and it is this feature that makes it naturally so suited for tasks that require coordination or synchronization. So remember I said there were two features of quantum entanglement.

So what is this second feature?

The second feature of quantum entanglement is that it's "**inherently private**".

If I have two qubits and they're completely entangled with each other, then it's physically impossible for any other qubit or actually anything else in the universe to have any share of this entanglement. This means that if I have a qubit it will be completely entangled with your

qubit. No one else can have a share of the entanglement or to entangle qubits form a private connection

WHY DON'T WE HAVE A QUANTUM INTERNET YET?

that no one else can share. So entanglement cannot be shared and it's inherently private. It's this feature that makes it naturally very suited for secure communication. Remember these two features.

It's **maximally coordinated** and **inherently private**.

So now I've told you all these things about qubits and entanglement.

But given that quantum entanglement is so cool you might ask why don't we have a quantum internet yet? It turns out that we can actually send qubits only over short distances.

You can go online and actually buy a commercially available box that performs quantum key distribution, Quantum secure communication over standard telecom fibre over distances of roughly 100 kilometres.

So the real challenge in building a quantum Internet is to get these qubits to travel further than these hundred kilometres.

Of course you might be asking why is it so difficult for these qubits to traverse long distances?

If I wanted to send a qubit down a communication line - we are sending a single photon - one particle of light.



Maximal coordination



You can imagine that if I take one single particle of light and I send it down a communication line, very soon it will be lost.

Remember that also qubits cannot be copied so if it's lost it's gone. I cannot resend and try again.

How can we hope to send these qubits over long distances so fortunately as I mentioned we can actually send them over short distances.

Let's put something like a box in the middle.



The box is not so far away from the left and it's not so far away from the right: namely close enough that I can send qubits both from the left and from the right to the box.

Let's try that. What we're going to do is to take two entangled qubits on their left and I'm going to send one of them to the box. The box is not far away, so we can do that. I'm going to take another two qubits which are entangled. I'm going to send one of them to the box.



Now we have entanglement with this box.

The cool thing is that there is a procedure called "**entanglement swapping**"* (see next page), with which we can glue this entanglement together and create entanglement over the entire distance! That's pretty cool.

So such a box is called a "**quantum repeater**" and we can use it to make entanglement over long distances. (See the schema next page - this includes some wider explanation). But you're probably saying what about this entanglement?

You promised that we could send my "Schroedinger cats", this particular data qubits that I want to send somewhere else.

There's a nice feature of quantum that if we have entanglement we can now send a data qubit by **teleporting** it across. So I take my yellow data qubit and I "teleport" it to the other side. This way we can send qubits over long distances.

We're probably saying this talk is getting more fantastical by the minute.

First there are these qubits now are we even going to teleport them around.

We are actually actively working on this box: the "quantum repeater" and by 2020 we want to have the first demonstration network in the Netherlands that showcases this box: the quantum repeater.

AMSTERDAM
LEIDEN
THE HAGUE
DELFT

It might become the first Quantum Internet in the world that actually connects small quantum computers, small quantum processors in such a way that we can send qubits from any of these quantum computers to another.

We can make entanglement between any of the two cities. You can sort of think that we are now on the edge of the quantum 1969 and in 2020 we want to send the first quantum message over what might be the first Quantum Internet



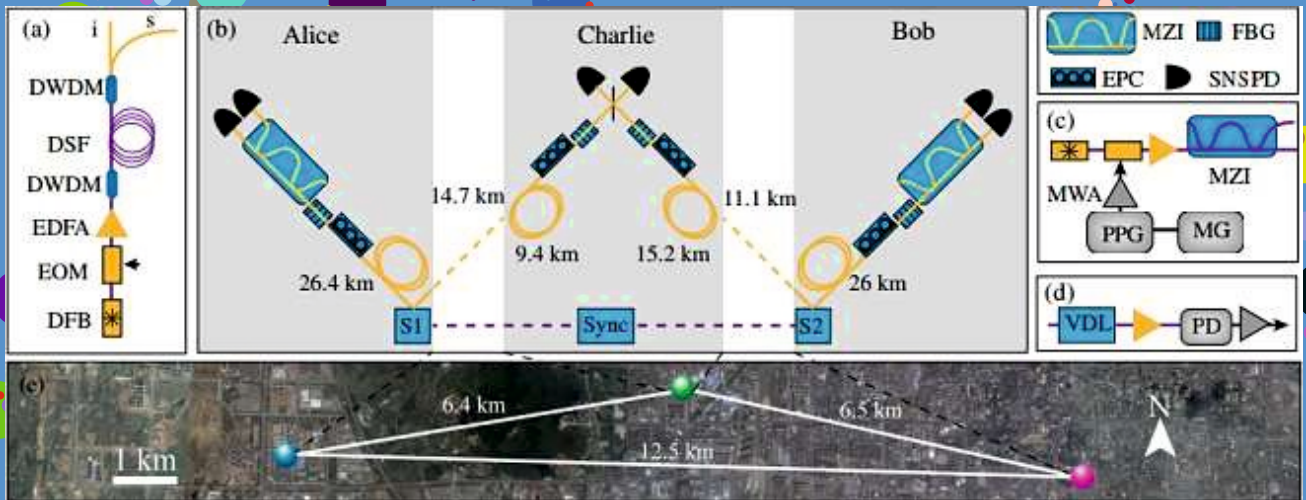


Figure above: this is an example of the schematic handling and just under it a map showing the distances



WIKIPEDIA

* In an important step for the infant field of quantum communications, researchers from the University of Geneva in Switzerland have, for the first time, realized an “entanglement swapping” experiment with photon pairs emitted continuously by two different sources.

Quantum teleportation is a process by which quantum information (e.g. the exact state of an atom or photon) can be transmitted (exactly, in principle) from one location to another, with the help of classical communication and previously shared quantum entanglement between the sending and receiving location.

Because it depends on classical communication, which can proceed no faster than the speed of light, it cannot be used for faster-than-light transport or communication of classical bits. While it has proven possible to teleport one or more qubits of information between two (entangled) atoms, this has not yet been achieved between anything larger than molecules.

Although the name is inspired by the teleportation commonly used in fiction, quantum teleportation is limited to the transfer of information rather than matter itself. Quantum teleportation is not a form of transportation, but of communication: it provides a way of transporting a qubit from one location to another without having to move a physical particle along with it.

A group of scientists led by Prof. Zhang Qiang and Pan Jianwei from the University of Science and Technology of China (USTC) have successfully demonstrated entanglement swapping with two independent sources 12.5 km apart using 103 km optical fiber.

Read more at:

<https://phys.org/news/2017-10-entanglement-swapping-independent-sources-100km.html#jCp>



starter

expert



INTRODUCTION

My name is Jean Pierre Hoefnagel and I founded a company called **1M2M** in The Netherlands. 1M2M designs end-to-end systems for geolocation and ultra-low power remote sensors using the **LoRa**, **Sigfox** and **NB-IoT** technologies. Very small packets of data (12, 16 bytes), long distance (30 to 100 km) and ultra-low power (several years on a single battery).

End to end means we develop our device hardware, radio and antenna circuits, firmware, test equipment, redundant server side database applications, websites and client specific Windows and web applications.

To build all parts of these systems a lot of things had to be developed. This article explains a very small but essential part of it.

This article describes a technique that has proven useful in several areas of our development, even if the applications seem to be completely different.

For the embedded systems we needed a way to 'decouple' different parts of the firmware. To make replacement and optional usage of drivers possible they cannot be directly linked to other parts of the firmware. However, there must be a 2-way communication.

For the PC-based systems we needed a bullet proof way for threads and **VCL** to interact. For our server applications we wanted a way to use a GraphicalUserInterface (GUI) on a different Windows machine than the Delphi Engine application.

Up until a few weeks ago, we had to send our customers Windows executables for client-side application software.

It would not run on android phones or Apple/Linux PCs. This caused a lot of trouble with firewalls, network issues, install-rights etcetera.

Since **TMS-Software** brought out **TMS Web Core**, there is a new application for this technique.

TMS Web Core makes it possible to run a quite complex user interface in any web browser, and still run the engine code in a normal Delphi application in a safe environment on one of our Windows servers. The GUI runs inside the browser and is connected to our servers by secure websockets.

The first thing you should know about threads is this: Threads. If you don't really, really need to use them then don't use them at all! It will save you a lot of headaches!

Having said that, if you cannot avoid them, you should at least make life easier by using a proven technology, especially if it can be put into a unit and you never need to think about it again. Our applications rely heavily on sockets to connect our servers to each other, websites, customer application software, devices in the field etc.

As most people know Indy sockets are blocking which make their use easy but if you do not want your **GUI** to block as well you will need to use threads to keep the **GUI** interactive.

For server sockets you don't even have a choice. Every client that connects to a server application opens its own service thread.

There are many mechanisms to share data between threads, and I tried a lot of them, but somehow as code got more complex nasty problems always started to arise.

Not the simple type of problems you can single step and debug, but the ones that cause a total application failure once every so many weeks. Until I implemented the following mechanism... Since then I have never seen these problems again .

As code grows more complex and gets divided over many units, it is not always obvious if a function is used by one of the many threads or by the **VCL** thread or even both.

For status monitoring, many functions or objects need to be able to show information in one or more locations in the **GUI**, without knowing how or where the **GUI** is implemented.

It seems a lot of extra effort in the beginning, but separation of **GUI** functionality and Engine functionality really helps maintaining the software later on, simply because it really helps to reduce developing spaghetti code.

This solution implements a messaging system. Any thread or **VCL** function can write information to it. Any other thread or **VCL** or even the same thread can read the information from it.



Messages can be sent and received from different units or objects in one application, but also between separate applications even when these applications are on different computers. This article describes messaging inside one application.

I will explain the remote (socket or websocket) implementation in another article.

I have written a working application to illustrate how this technique works.

Please feel free to experiment with it and extend it to your own needs.

The complete source code is available at Blaise Pascal Magazine as **BPM_MessageDemo.zip**.

For the full demo there are 4 Pascal source files.

- A thread object that can be used as ancestor of message-aware thread objects in **uBPM_BaseThread.pas**.
- A basic set of message functions in **uBPM_Messages.pas**
- An example of message-aware thread object in **uBPM_Task1.pas**
- The demo MainForm (GUI) is defined **fBPM_MessageDemo.pas** and **fBPM_MessageDemo.dfm**.

In the application I use some other special functionality like singletons and colored screen logs. They will be explained briefly where needed.

THE TBPM_BASETHREAD OBJECT (UNIT UBPM_BASETHREAD.PAS)

```
type TBPM_BaseThread = class(TThread)
private
...
protected
...
public
  constructor Create();
  function OKToRun: boolean;
  procedure OnMaint(var Done: Boolean); virtual;
  procedure OnMaint1000(); virtual;
  procedure OnActivate(); virtual;
  procedure OnBeforeExecute(); virtual;
  procedure OnAfterExecute(); virtual;
  procedure OnDeActivate(); virtual;
  procedure OnMsgHandler(mr:TBPM_MessageRec); virtual;
  procedure OnErrorlog(Cde:integer;Msg:string;Color:TColor);
virtual;
  property Active:boolean read fActiveAck write fActiveReq;
  property MsgID:TBPM_MessageID write SetMsgID;
  function GetCPULoad: double;
end;
```


When you look at this thread object, you will probably miss the `Execute()` function. It is replaced by the `OnMaint(var Done:boolean)` procedure. This procedure is called repeatedly while the `Active` property is true.

Do not place an endless `while do` in this function, just do (part of) a task and return. For processes that use a lot of time it is advised to use a state machine to break up the operation into smaller parts.

If the `var` parameter **Done** is set to `False` the next call is made as soon as possible, if not the **CPU** load is reduced to close to zero by executing a `sleep()` instruction every loop.

There is also a `Maint1000()` function which is called every 1000 ms while the `Active` property is true for common tasks like updating **GUI** info every second.

Once created the thread itself keeps running until the application closes. For this the main form has to set the global variable `EndAllThreadsRequest` to true in the evenhandler `FormCloseQuery()`. All BPM thread descendants will terminate and free automatically.

The property `Active` controls whether the `OnMaint()`, `OnMaint1000()` and `OnMsgHandler()` procedures get called or not. Any exceptions in these three procedures will be handled by calling the `OnErrorLog()` function with information about the cause of the exception.

For lengthy operations or loops in these three procedures, it is advised to check for the `OKToRun()` function. If it returns false, the application is shutting down and is probably waiting for this thread to exit.

Messages will not be handled anymore because other threads or **GUI** code might already have shut down.

Changes in the `Active` state cause a call to `OnActivate()` and `OnDeactivate()` handlers. Their main purpose is to make the GUI show the thread state like a "Led" indicator in an "Active" button or so.

`OnBeforeExecute()` and `OnafterExecute()` get called when the thread first starts and when the thread is terminated. The main purpose is to construct and destruct used objects like you would normally do in the `Formcreate()` and `Formdestroy()` procedures.

All procedures described so far run from the thread. Be careful not to access **VCL** procedures from within them!

The `Active` property can be set or cleared by a **VCL** function. Reading it back gives the actual value for the running thread.

If a thread object needs to receive messages you can setup your own message handler(s) in `OnMaint()`, but you can also use the built in message handler. To do this you will have to set the `MsgID` property to a value -1 or higher. -1 means all messages are received, 0 or higher means only that message ID is received.

A value of -2 or lower will disable the message receiver. Message IDs are described a little further in this article. Received messages are passed to the objects `OnMsgHandler()` procedure as a `TBPM_MessageRec` record. The constructor `Create()` takes no parameters, which is a necessity for using it in Singleton objects (also described later).

If any of the procedures is not used, simply do not override them. The base class has empty dummy procedures for that.

Finally a `GetCPULoad()` function is provided. It can be called from any thread including the VCL and returns the **CPU** Load of this thread object in 0.100 percent. Useful for status monitoring. This function needs the `WinAPI`, for use in Lazarus please remove or replace it.

THE MESSAGE FUNCTIONS

(unit `uBPM_Messages.pas`)

The message system is fully thread safe. It has one or more subscribers that register their own `FIFO` message queue with or without `MsgID` filter.

When a procedure sends a message, it is put into all available message queues if their filter matches the message ID or is disabled.

Each subscriber polls its queue and retrieves as many messages from it as possible.

To prevent memory errors a queue is limited to 10.000 entries, otherwise the queue would keep growing if a subscriber fails to maintain it.

The interface section of the unit defines these types and functions:

```
type TBPM_MessageID = integer;
type TBPM_CommandID = integer;

type TBPM_MessageRec = record
  MsgID: TBPM_MessageID;
  Cmd : TBPM_CommandID;
  Msg : string;
  Color: TColor;
end;

function BPM_MsgBufOpen (aFilter:TBPM_MessageID):THandle;
procedure BPM_MsgBufClose (aHandle:THandle);
procedure BPM_MsgBufWrite (aMsgID:TBPM_MessageID; aCmd:TBPM_CommandID;
  const aMsg: string; aColor: integer);
function BPM_MsgBufRead (aHandle:THandle;var aMsgRec:TBPM_MessageRec):boolean;

function BPM_RegisterMsgID(const MsgIdName:string):TBPM_MessageID;
function BPM_RegisterCmdID(const CmdIdName:string):TBPM_CommandID;
```

The actual message system uses only four functions.

- **BPM_MsgBufOpen()**
for setting up a message queue and obtaining a handle to it.
- **BPM_MsgBufClose()**
for closing a message queue and freeing all associated memory.
- **BPM_MsgBufWrite()**
for writing messages to the system.
- **BPM_MsgBufRead()**
for reading from the message queue that belongs to the handle.

There are two helper functions that can be used for obtaining unique integer `MessageIDs` and `CommandIDs`, but if you like to define these IDs as constants it will also work fine.

To set up a message receiver a few things have to be coded.

1. Obtain a message ID, either name-based by calling `BPM_RegisterMsgID()` or by defining an integer constant in a shared source file.
2. For any receiver that has to read from this `MessageID`, register a handle by calling `BPM_MsgBufOpen()`.
3. In a VCL form, use the `ApplicationEvents.OnIdle()` hook to call your `OnIdle()` procedure. In a thread object, just call your `OnIdle()` function from within a loop in the `OnExecute()` function.

For the transmitter it is even simpler:

1. Obtain a message ID, either name-based by calling `BPM_RegisterMsgID()` or by defining an integer constant in a shared source file.
2. Call `BPM_MsgBufWrite()` directly, or use a more convenient `log()` procedure to send messages.

Typical use of the `BPM_MsgBufWrite()` function would be like this:

```
constructor Tfrm1.FormCreate(Sender: TObject);
begin
    MsgID_GUI := BPM_RegisterMsgID('BPM_MainLogWindow');
    MsgID_Task1 := BPM_RegisterMsgID('BPM_Task1');
end;

procedure Tfrm1.Log(Cmd:integer; Msg:string; Color:TColor);
begin
    BPM_MsgBufWrite(MsgID_GUI,Cmd,Msg,Color);
end;

procedure TfrmBPM_MessageDemo.edTextChanged(Sender: TObject);
begin
    BPM_MsgBufWrite(MsgID_Task1,Task1.cmd_OnEDTextChanged,edText.Text,0);
end;
```

The Create sets values for the `MsgID_GUI` and `MsgID_Task1`. `MsgID_GUI` is used to send data to the **GUI**, and, as you could expect, `MsgID_Task1` is used here to send messages to a thread called `Task1`. For writes to the **GUI** the procedure `Log()` is used, Messages to `Task1` can be sent directly from the **GUI** event handlers.

Typical use of the `BPM_MsgBufRead()` procedure would be like this:

```
constructor Tfrm1.FormCreate(Sender: TObject);
begin
    MsgHan := BPM_MsgBufOpen(-1); // -1 means listen to all messages,
    // not a specific MsgID
end;

destructor Tfrm1.FormDestroy(Sender: TObject);
begin
    BPM_MsgBufClose(MsgHan);
end;

procedure TfrmBPM_MessageDemo.OnIdle(Sender: TObject; var Done: Boolean);
var MR:TBPM_MessageRec;
begin
    while (BPM_MsgBufRead(MsgHan,MR)) do OnMessage(MR);
end;
```

The `FormCreate()` opens a receiving message queue and stores its handle in `MsgHan`. If the parameter is 0 or higher, only messages are queued with that message ID.

A value of -1 selects all sent messages, regardless of the `MsgID`. In that case `MsgID` dependent behavior must be implemented in the message handler. The `OnIdle()` procedure calls `OnMessage()` with new messages for as long as messages are in the Queue.

The `FormDestroy()` frees all memory associated with this message queue.

THE DEMO APPLICATION (unit `fbPM_MessageDemo`)

Now, let's build a demo application with this knowledge!

There are 2 basic modules. One that implements a form for the **GUI** part, and one that defines a thread object.

The `TTask1` thread object is derived from the `TBPM_BaseThread` class in the `uBPM_BaseThread` unit. `TBPM_BaseThread` publishes eight procedures which make our thread -life a little easier. They can be overridden by descendants when needed.

The most important ones are:

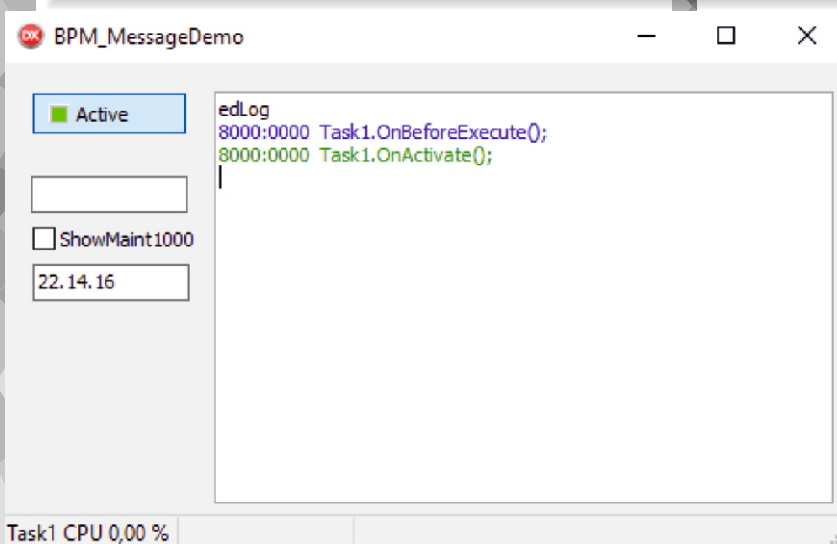
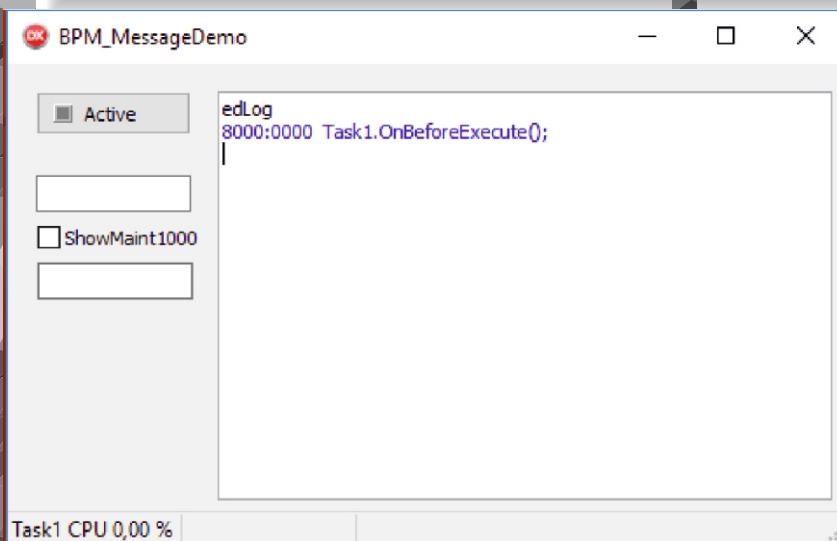
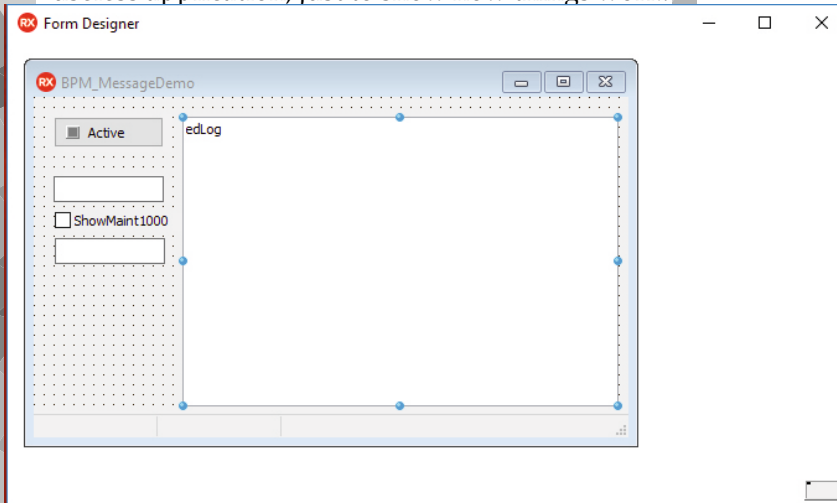
- `OnBeforeExecute()` Use this to create and initialize used objects
- `OnAfterExecute()` Use this to free used objects
- `OnMaint()` Called as often as possible while **Active** is **True**
- `OnMaint1000()` Called every second while **Active** is **True**
- `OnMsgHandler()` Called when a message is waiting and **Active** is **True**

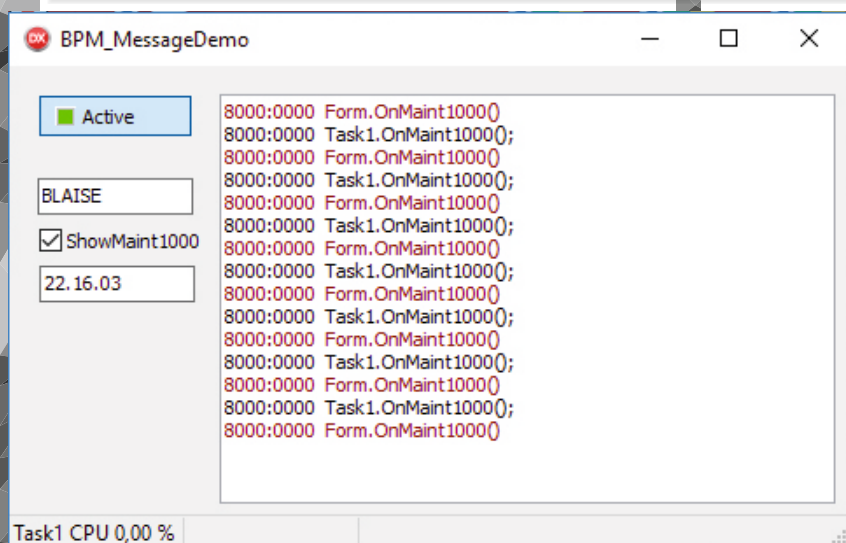
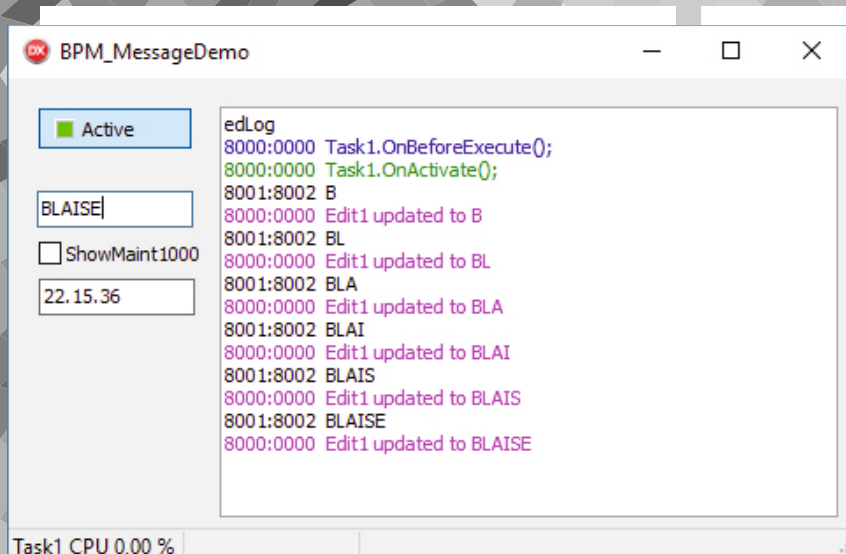
In the GUI module (The form) 5 similar procedures are implemented.

- `OnInit()` Called after all constructors and inits on first **FormShow()**
- `OnDone()` Called just before form closes, on **FormCloseQuery()**
- `OnMaint()` Called as often as possible by **TApplicationEvents.OnIdle()**
- `OnMaint1000()` Called every second by a **TTimer** instance
- `OnMsgHandler()` Called by **OnIdle** whenever a new message is waiting

These 5 procedures run the whole application, both for Thread objects and for Form objects.

For this demo, I wrote a small and absolutely useless application, just to show how things work.





There is a `TSpeedbutton` called `btnActive` with a kind of LED-indicator to activate and deactivate the thread object. The `btnActiveClick()` method code does not set the led color, it only sets the thread property to true or false.

The Led color is set by a message sent by the thread object.

There is an `TEdit` for inputting text. Any change is sent to the thread by its `edTextChanged()` method. The thread responds by logging an updated value to the logwindow in purple.

There is a `TCheckbox` to disable or enable an action in the `Maint1000()` of both the form and the thread object. The **GUI** uses the `Checked` property directly to enable or disable messages from the `OnMaint1000()` method.

The `cbShowMaint1000Click()` also sends the checked value to the thread, which uses it to enable or disable its own `maint1000()` logging. There is a read-only `TEdit` object to show some info sent by the thread, in this case the time. Finally, there is a `TRichEdit` component to show log info in different colors.

What happens is that all controls broadcast their status changes and or values to a message that can be received by anyone that listens to it.

Any listener can send messages that will set the values or status of these **GUI** controls.

Be carefull to temporarily disable the `OnChange` events when setting a value or property from the message handler.

Note: Otherwise the change will trigger a new message and the system can get very busy!

Listeners for status and value updates should also listen to messages from other objects that can set values to stay in sync.

THE DEMO APPLICATION

(unit `uBPM_Task1`)

This object is implemented as a 'singleton'.

This means the thread object is not referenced by one or more variables in a form or other objects, but has a single reference.

It is created automatically when another object uses it, and disposed of in the finalization section of the unit or in this case by the

`FreeOnTerminate()` of the thread.

Implementation of a singleton can look like this:

```
unit uBPM_Task1;

interface uses ...

type
  TTask1 = class(TBPM_BaseThread)
  private
    ...
  public
    ...
  end;

function Task1():TTask1;

implementation

var fTask1:TTask1;
function Task1():TTask1;
begin
  if not assigned(fTask1) then fTask1 := TTask1.Create();
  result := fTask1;
end;

... (object code)

initialization
  fTask1 := nil;
finalization
end.
```

To access it, just use the function `Task1()` as if it were a variable. If the object does not exist (yet) it is created, otherwise, the existing object is used.

If used this way, only one instance will be created and used, even if many other objects make use of it.

This is actually the simplest unit in this **demo project**. It registers used message IDs and commands in its `Create` function, This can also be done in the `Init()` function or in the units `initialization` section. Then it uses these IDs and commands to setup a listener, and to send messages from all if its `OnSomething()` methods. The `OnMsgHandler()` function sets local variables and sends log information to the GUI-object.

The project code is a working but simplified version of our existing library code. It just contains the basic elements to explain the principles. To add binary messages, just overload the message functions and add different types of `TBPM_MessageRec` records according to your needs. Please read the code, it is pretty easy to see what happens, and start playing with it will help you see its use for thread or socket applications. Hopefully this article and the included project code are useful to help you write simple and stable threaded code.

If there are remarks or improvements, please contact Detlef Overbeek.

Enjoy !

Jean Pierre Hoefnagel, **1M2M B.V.**

DELPHI CONFERENCE

2018

DEVELOP YOUR FUTURE

JAARBEURS UTRECHT
18 SEPTEMBER 2018

barnsten

PASCON

 embarcadero

DELPHI CONFERENCE 18 SEPTEMBER 2018

08:30-09:30 Welcome and registration with coffee and tea

09:30-10:30 **KEYNOTE: MARCO CANTÚ - PRODUCT MANAGER DELPHI - "DELPHI 10 FOR WINDOWS 10 AND BEYOND"**

In this technical keynote, Marco will cover the status of Delphi 10 and what's coming, with a particular focus on Windows 10 support, VCL development for Windows 10, but also covering Delphi mobile and server solutions and the overall industry trends the product is part of

10:30-11:30 **BRIAN LONG - CREATIVE DELPHI DEBUGGING TECHNIQUES**

Debugging represents a big part of development, perhaps one of the biggest. We all know about breakpoints, single-stepping and watches, but what else can we do to help work through bug scenarios and resolve problems?

This session looks at a number of techniques, tricks, and utilities to help make the chore of debugging a bit more productive. Warning: this session may contain the CPU window!

11:30-11:50 **Coffee Break / Go to Breakout Sessions**

11:50-12:40 **Brian Long**

HOW TO ACCESS THE ANDROID API

The ability to build Android applications is a great aspect of recent versions of Delphi, which gets more capable and functional with every release. However exploring outside the "FMX envelope" is still an onerous task to all but the most propeller-headed of Delphi developers.

We'll look at how to pull in various "not-in-the-box" features into an Android application using the latest version of Delphi and hopefully take away the mystery associated with it.

11:50-12:40 **Bruno Fierens**

A RADICALLY NEW WAY TO DEVELOP MODERN WEB APPLICATIONS

The all new TMS WEB Core product brings exciting new ways to create modern, fast and responsive web applications using the SPA model. This enables Delphi devs to use the familiar Delphi language and RAD development techniques to create web apps directly from the IDE. While TMS WEB Core facilitates creating the UI logic completely with Delphi using a Pascal to JavaScript compiler, the framework is extensible to consume popular JavaScript libraries and frameworks such as Bootstrap, jQuery, etc... TMS WEB Core also empowers Delphi developers to leverage the TMS FNC UI Controls framework as UI controls for web applications, reusing the VCL or FMX UI logic.



TMS WEB Core
Framework for creating modern web applications

12:40-13:30 **Lunch - Go to Break Out Sessions**

13:30-14:20 **Roald van Doorn**

CONTINUOUS DELIVERY WITH EXISTING VCL APPLICATIONS

A case study of how we applied CD principles to an older VCL application. We will take a look at the challenges we faced and the solutions we chose, the frameworks we use, release procedures and feedback loops. We will demonstrate how we safely build and deploy the Windows software for Albelli en Vistaprint and the benefits this brings to our team and organization. Outline: automating your builds using TeamCity - Unit testing using DUnitX - Automated UI tests using Ranorex and Specflow - Deploy to different environments with ProGet and Octopus Deploy - Increase speed of value to customer (reduced stock) - Increased feedback to developers.

13:30-14:20 **Daan van der Werff**

DELPHI OP DE WERKVLOER "GROOTHANDEL & MAGAZIJN"

Tijdens deze sessie krijgt u een kijkje onder de motorkap van een groothandel waar kritische processen gemaakt zijn in Delphi. Deze zijn verantwoordelijk voor een omzet van ca 31 miljoen! Van data connectoren tot orders, microservices, mobile en cross platform ontwikkelingen voor warehouse management systemen en meer!

https://www.barnsten.com/default/events/details?events_id=327

DEVELOP YOUR FUTURE

Delphi Conference 2018 Jaarbeurs Utrecht Netherlands

14:20-14:30 Go to next Break Out Sessions

14:30-15:20 **Danny Wind**

MICRO SERVICES AND PROGRESSIVE WEB APPS (PWA) DELPHI

In this session we'll showcase a lightweight REST microservice and a (progressive) web app, as well as an Android/iOS App and a desktop application all crated in Delphi. With the techniques in this session you'll be able to leverage these new technologies in your own projects. Just re-use the sources and you're ready to go.

14:30-15:20 **Bob Swart**

DELPHI EN FIREDAC ENTERPRISE CONNECTORS

De FireDAC Enterprise Connectors stellen Delphi ontwikkelaars in staat om externe data bronnen beschikbaar te maken als (FireDAC) tables en queries, voor gebruik en verwerking met FireDAC data-access componenten. In deze sessie zal Bob de algemene werking van de FireDAC Enterprise Connectors laten zien, met veel code voorbeelden, en daarbij een aantal specifieke toepassingen demonstreren met externe bronnen zoals bijvoorbeeld Facebook, Twitter, LinkedIn maar ook Gmail, Google Drive, Google Analytics en een generieke REST en JSON connectie.



15:20-15:40 Break

15:40-16:30 **André Mussche**

DE OPKOMST VAN SPRAAKHERKENNING

André werkt momenteel met het nieuwe realtime en streaming protocol gRPC dat vrij recent door Google is ontwikkeld. gRPC wordt bijvoorbeeld gebruikt bij Blockchain implementaties zoals hyperledger, maar is ook uitermate geschikt voor de toepassing in projecten met spraakherkenning. Het gebruik van spraakherkenning in applicaties wordt steeds meer toegepast en wordt bijvoorbeeld in ziekenhuis applicaties veel gebruikt. Maar ook in ERP systemen wordt dit steeds vaker toegepast. In deze sessie krijgt u te zien hoe u met dit communicatieprotocol een extra dimensie kunt toevoegen aan uw applicatie met het door André ontwikkelde protocol voor Delphi toepassingen dat inmiddels ook als open source beschikbaar is.

15:40-16:30 **Marco Cantú**

RAD SERVER IN DEPTH

This session offers a deeper look into the development of REST + JSON web services with RAD Server, going beyond the basic marketing information and introductory demos, and highlighting some advanced features like dynamic resources and custom login modules, recent web and JavaScript support additions, touching on ExtJS clients, and providing indication of new coming features.

16:30-17:00 Closing - Q & A - Prize Draw

https://www.barnsten.com/default/events/details?events_id=327

DEVELOP YOUR FUTURE
Delphi Conference 2018 Jaarbeurs Utrecht Netherlands

INTRODUCTION

Games are a powerful medium to tell stories. They can be as interactive as we want, and they can freely gather ideas from other arts. Books, music, graphics, they all can be mixed into something wonderful. If you ask a game developer why did (s)he chose this career, the answer is often a beautiful game (s)he once enjoyed. For me, this game was Wizardry 7. An epic RPG game for DOS, with a magic and dangerous world, and a grand tale to uncover. I spent half a year of my life playing this game, and never had any regrets about it. Now if you ask a game developer how did they manage to actually get into this career, they will often tell a story about a game studio they formed together with a friend. They envisioned a game they want to make, they signed a contract with a publisher, they downloaded and learned the necessary tools and got to work.

Personally, I did not receive the memo that you can get some ready tools to make games. Instead, I was fascinated by the free software and open-source movement around Linux, and I had this bold feeling that I can code everything I want in Pascal from scratch. So when I wondered "how to make a game", I thought: "Well, it should be simple.

I can take this variable and call it **PlayerHealth**. And another one, and it will be **MonsterHealth**. Now if you put them both together in the same room...". I knew I want to make my own tools to create games, and I want them to be open-source. That's how Castle Game Engine was born. Years later, we now have our own game studio, **CAT-ASTROPHE GAMES**. We're making games using Castle Game Engine, and develop the engine as an open-source project, for everyone to benefit.

ADVANTAGES OF CASTLE GAME ENGINE

We have a lot of documentation on our website

<https://castle-engine.io/>,
in particular we have a large manual for developers:
https://castle-engine.io/manual_intro.php.

You can create all kinds of games. 3D or 2D, in Blender or Spine or any other authoring tool. We can load many file formats. Our website contains a section devoted to Creating Game Data where we document the details:

https://castle-engine.io/creating_data_intro.php

The engine is cross-platform, we support many desktop (Windows, Linux, macOS) and mobile (Android, iOS) platforms. We use modern OpenGL and OpenGL ES rendering, planning also other renderers (Vulkan) in the future. We support a number of services on mobile platforms to integrate your game with native system features, like analytics, in-app purchases and more.

The engine is not limited to developing games. You can use it to make any crossplatform application where visualization of some 3D or 2D stuff is important. For example you can make a 3D editing application, or a visualization of some industrial machinery. We use an international standard called X3D to define our scene graph. Absolutely everything that you load from 3D files can also be created or modified by your Pascal code.



The engine has own user interface system. The system supports anchors and smart scaling (to adjust to any screen resolution). The look of all the controls is configurable, which is often a necessity for games, where you want the user interface to match the look of your game.

The engine control can also be placed inside a Lazarus form, and surrounded by regular Lazarus (LCL) user interface.

The open-source nature of the engine is a big advantage too. The engine is written using a clean modern Object Pascal language, and you use it to create games in Pascal as well.

Likewise, you can use our game assets, or you can create your own e.g. in Blender. Start by downloading the **Castle Game Engine** from <https://castle-engine.io/>.

This is a large zip file with the engine source code. Unpack it anywhere on your hard disk. You will also need the latest version of FPC and Lazarus, that you can get from <http://www.lazarus-ide.org/>.



So every user is a potential engine developer. If you want to tweak the engine to your private needs — you can do it. And if your modification improves the engine for everyone, we encourage you to send us a pull request.

DOWNLOADING AND INSTALLING

Let's jump in! We will make a simple 3D game using Castle Game Engine from scratch.

The game data (models and textures), as well as the final source code, is available online on <https://github.com/castle-engine/blaise-pascal-article-examples>.

You can follow this article to create your own source code, or you can just download a ready application from this repository.

Compile and Install Lazarus Packages

Within Lazarus:

1. Compile (but do not install) the package `castle_game_engine/packages/castle_base.lpk`.
2. Compile (but do not install) the package `castle_game_engine/packages/castle_window.lpk`.
3. Compile and install the package `castle_game_engine/packages/castle_components.lpk`.

It will actually also install `castle_base.lpk` (as a dependency), which is OK.

See the

<https://castle-engine.io/documentation.php>

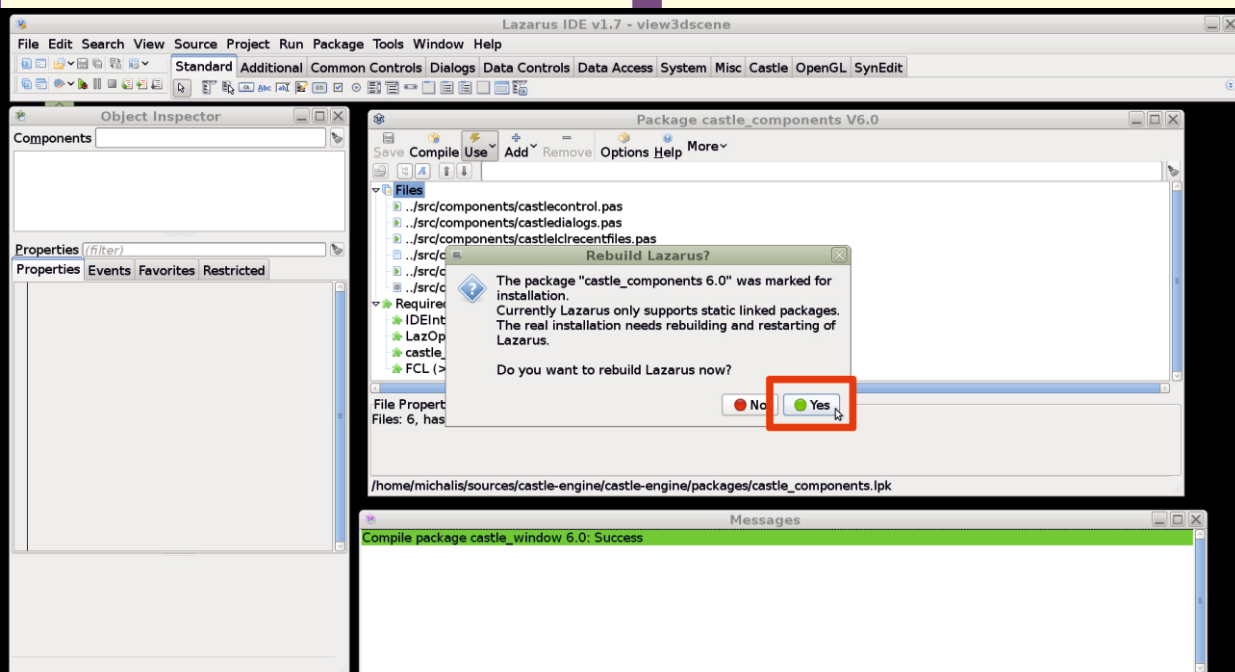
for screenshots and detailed instructions how to do it.

Build Tool and the Upcoming Editor

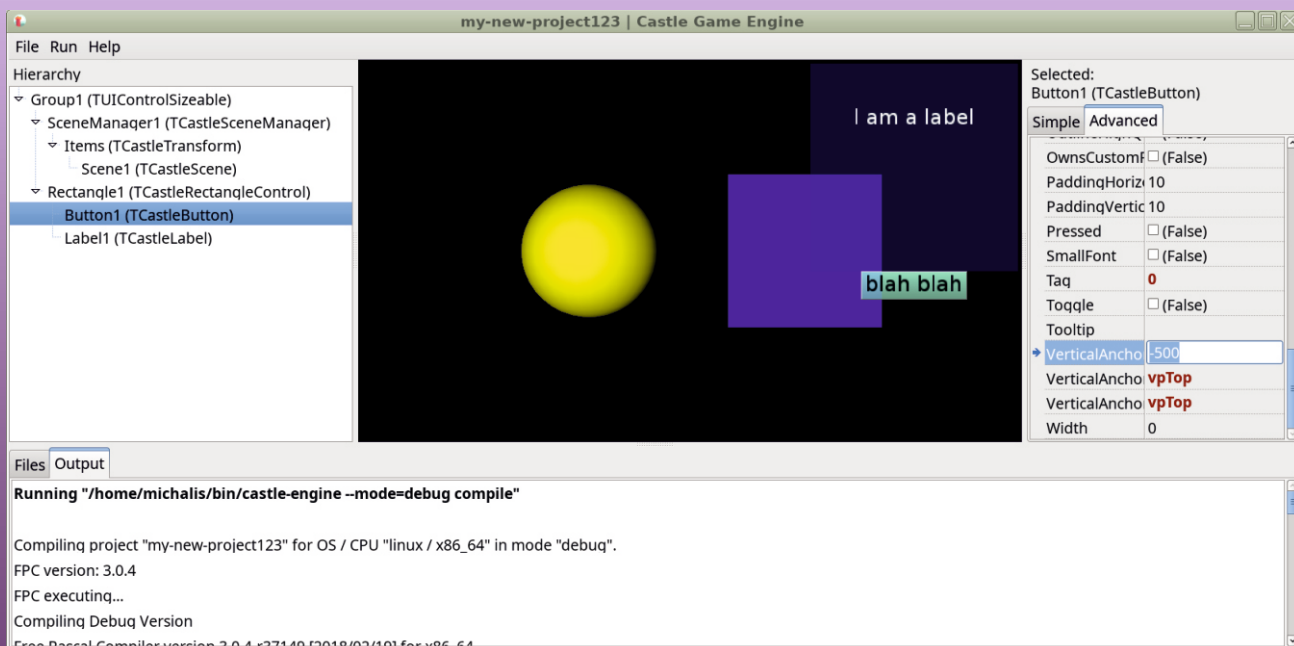
The engine can also be used without Lazarus or LCL. We have our own build tool (called **castle-engine**) that can compile and package cross-platform applications. In particular, it can create a ready-to-install Android `.apk` file for your game. The build tool calls FPC and other tools under the hood.

We will describe how to use the build tool to compile our demo game for Android in the later section "Using the Build Tool to Compile for Android".

We are also working on **Castle Game Engine Editor** that allows to manage your projects and design the user interfaces and 3D and 2D game worlds visually. But it is not stable yet, so we will not use it in this article.



Next, I encourage you to test some of the engine numerous examples. I advice testing **examples/fps_game/** (a complete game using the engine) and **examples/lazarus/model_3d_viewer/** (the engine control inside a Lazarus form). Simply open their project files (`.lpi`) in Lazarus and run them.



CHOOSING HOW TO CREATE A WINDOW

You can use Castle Game Engine in two ways:

1. Drop the TCastleControl visual component on a Lazarus form. This is great if you want to use Lazarus to create natively-looking user interface, and only use Castle Game Engine to add some rendering inside a form. The TCastleControl is basically an OpenGL context with a lot of Castle Game Engine features for rendering and input added.
2. Use TCastleWindow class as a container for the Castle Game Engine rendering and input. This means that you don't use Lazarus forms and LCL at all. All the user interface is done using Castle Game Engine. TCastleWindow is a window containing only OpenGL (or OpenGL ES) context. This is a better choice if you don't need natively-looking user interface, and prefer an interface custom to your application. This is a common approach for games. *

*(*TCastleWindow actually features natively-looking menu bar and dialog boxes. But that's it. Everything else is custom-drawn using OpenGL(ES) and Castle Game Engine.)*

The main advantage of TCastleWindow is that it works on all platforms, including mobile (Android and iOS). Mouse events (dragging and mouse look) are also processed smoother by our TCastleWindow, thanks to our own event loop. You can of course still use Lazarus as an IDE, to write and debug code.

You will however not use Lazarus form designer in this case. This article will show the TCastleWindow method to initialize the window. If you are unsure what to choose, don't sweat it. The Castle Game Engine API is the same in both cases, and changing your mind later is not hard.

The repository on <https://github.com/castle-engine/blaise-pascal-article-examples> contains two versions of our game — one using TCastleWindow (in 3d_game subdirectory) and the other using TCastleControl (in 3d_game_alternative_using_castlecontrol subdirectory).

DEVELOPING A SIMPLE 3D GAME

OPEN A WINDOW

This is the final version of our game, running on desktop and Android:



Start by creating a new project in Lazarus that uses `castle_window` package, and doesn't use LCL (Lazarus Component Library):

- Create a new project using Lazarus "File → New ..." menu item.
- Choose "Project → Simple Program" (or "Custom Application" in older Lazarus versions).
- Using the "Project → Project Inspector" window add a "New Requirement" and choose `castle_base` package. Then add another new requirement and choose `castle_window` package.
- Save your project using the menu item "Project → Save Project ..." under any name, like `my_game`.

Your application source code is now a single file `my_game.lpr`. It looks like this:

```
program my_game;
uses castle_base, castle_window;
begin
end.
```

To open a window, simply construct an instance of the `TCastleWindow` class. Then call the method `OpenAndRun` to open (show) the window and run the application.

To do this, use unit `CastleWindow`. You can also remove the package units `castle_base`, `castle_window` from your `uses` clause, they are not really necessary. Here's the final code:

```
program my_game;
uses CastleWindow;
var
Window: TCastleWindow;
begin
    Window := TCastleWindow.Create(Application);
    Window.OpenAndRun;
end.
```

You can run it and be amazed by an empty black window that appears.

On Windows, you will notice an additional console window that appears each time you run the application. To avoid it, go to the "Project → Project Options ..." dialog in Lazarus, and check the option „Win32 GUI Application" on the "Compiler Options → Config And Target" page.

Alternatively, add

```
{$ifdef MSWINDOWS}
{$apptype GUI}
{$endif}
```

to the source code.

Open a Window in a Cross-Platform Way

To make the application cross-platform, portable also to mobile platforms, you should move the window creation into a separate unit.

The convention used throughout the engine examples is to call this unit `GameInitialize` or even just `Game`. The window should be created in the unit's initialization section and assigned to a special property `Application.MainWindow`.

Note that cross-platform applications are limited to using a single window, the one you set in `Application.MainWindow`. While on desktops you can create as many `TCastleWindow` instances as you want, you don't have this luxury on other platforms.

The main program file only needs to call `Application.MainWindow.OpenAndRun`; now. On mobile platforms, the main program file will not be used. Instead, a special library will control the creation and display of the game window. This library will only include your units (`GameInitialize` and anything else you use) to initialize and run the game.

More details about compiling for mobile are available in the later chapter "Using the Build Tool to Compile for Android" of this article. For now let's only make sure we are ready for this. The new version of our application is split into two files.

1. The unit `GameInitialize` in file `gameinitialize.pas` with this content:

```
unit GameInitialize;

interface

implementation

uses CastleWindow;

var
    Window: TCastleWindow;

initialization
    Window := TCastleWindow.Create(Application);
    Application.MainWindow := Window;
end.
```

2. The program in file `my_game.lpr` with this content:

```
program my_game;

program my_game;
uses CastleWindow, GameInitialize;
uses CastleWindow, GameInitialize;
begin
    Application.MainWindow.OpenAndRun;
end. Application.MainWindow.OpenAndRun;
```

You can run the new version of the application. It will work the same as before (just show an empty window), but now it's a cross-platform empty window. For more information about creating cross-platforms games see the documentation on https://castle-engine.io/manual_cross_platform.php.

Load 3D Model

You're probably anxious at this point to display some cool 3D object using the engine.

Let's do it.

First we need to get some sample 3D model.

You can use a sample model of a soldier from https://github.com/castle-engine/blaise-pascal-article-examples/tree/master/3d_game/data/character.

Get the model in `.castle-anim-frames` format from there. It is easiest to just download the complete repository <https://github.com/castle-engine/blaise-pascal-article-examples/> and take the whole `3d_game/data/character` subdirectory from it.

To load a 3D model, you create an instance of `TCastleScene` class and use the `Load` method on it. `TCastleScene` is probably the most important class in the entire Castle Game Engine, responsible for loading, animating and rendering game models.

Where should you load this? In applications using `CastleWindow`, it is best to perform the initialization of your game inside a callback assigned to the `Application.OnInitialize`.

This callback is always run once, as soon as the engine is ready (all internal resources are initialized and the drawing context is ready), and before calling any other engine event. Note that there are many other possibilities.

In a Lazarus LCL application, you could create your model inside the `TForm.OnCreate` or `TForm.OnShow` event.

In principle, you can create the `TCastleScene` and load it at any point during your game. In a cross-platform application you should only be careful to never load files before the

`Application.OnInitialize` was called (in particular, do not load files inside the unit's initialization clause). Here's a working unit loading and displaying a 3D scene:

```
unit GameInitialize;

interface

implementation

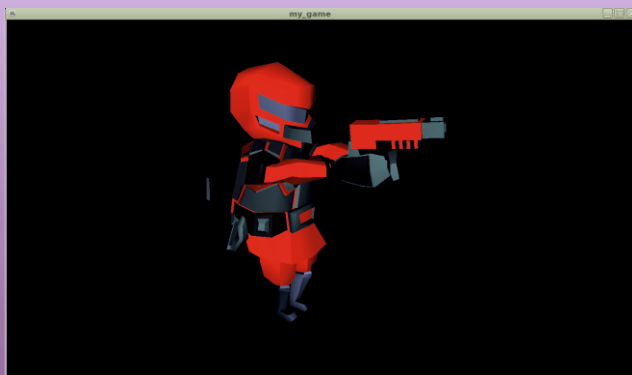
uses CastleWindow, CastleScene;

var
  Window: TCastleWindow;
  SoldierScene: TCastleScene;

procedure ApplicationInitialize;
begin
  SoldierScene := TCastleScene.Create(Application);
  SoldierScene.Load( 'data/character/soldier1.castle-anim-frames' );

  Window.SceneManager.Items.Add(SoldierScene);
  Window.SceneManager.MainScene := SoldierScene;
end;

initialization
  Window := TCastleWindow.Create(Application);
  Application.MainWindow := Window;
  Application.OnInitialize := @ApplicationInitialize;
end.
```



What is a Scene Manager?

The above code also deals with something called `SceneManager`, an instance of the `TCastleSceneManager` class.

Scene manager is a 2D user interface control that by default acts as a viewport (rectangular area through which you see a 3D or 2D world) and a central keeper of information (what is contained in this world).

If you use `TCastleWindow` class (not just `TCastleWindowCustom`) then a default full-screen scene manager is already created for you and available in the `Window.SceneManager` property. A default scene manager is empty, and simply draws a black background.

In our above example, we have:

1. Added the scene to `SceneManager.Items`, to make it actually visible.
2. Set the scene as `SceneManager.MainScene`, to make it determine the initial headlight.*

*(*In new Castle Game Engine 6.5, we could also set `SceneManager.Headlight := hlOn`, there's no need to set `SceneManager.MainScene` only for this purpose.)*

More complicated games often deal with scene manager in a more complicated way.

- You can create instances of `TCastleSceneManager` yourself, as many as you need, and show and hide them as necessary. You can avoid creating the default scene manager by using `TCastleWindowCustom` instead of `TCastleWindow`.
- By default scene manager is both a viewport and a keeper of information. But you can add additional `TCastleViewport` instances (that refer to a central `TCastleSceneManager` instance) to display multiple views (from multiple cameras) of the same world. In this case you can also stop the scene manager from acting as a viewport by setting `SceneManager.DefaultViewport` to `false`.
- By default scene manager in opaque, filling the window with a black color underneath everything is renders. You can change it by changing the background color (`SceneManager.BackgroundColor := Vector4(1, 1, 0, 1);` or `SceneManager.BackgroundColor := Yellow;`), or making the scene manager transparent (`SceneManager.Transparent := true`).
- Just like all other user interface controls, you can change the size and placement of the scene manager. It doesn't have to fill the whole screen.

Where to Keep Your Data?

It is best to place all your game data (models, textures, and everything else you may load) inside the data subdirectory of your project. This directory is a little special, as it will be automatically correctly packaged by the build tool and available for your application on all platforms, including mobile Android and iOS. On Unix, the application data may also be installed system-wide.

We have a special function `ApplicationData` (in the `CastleFilesUtils` unit) that allows to refer to the data files inside your project in a cross-platform, customizable way.

Instead of

```
SoldierScene.Load('data/character/soldier1.castle-anim-frames');
```

you can write this:

```
SoldierScene.Load(ApplicationData('character/soldier1.castle-animframes'));
```

This will work on all platforms.

Note that `ApplicationData` is available in the unit `CastleFilesUtils`. You may need to add it to your uses clause.

Using `ApplicationData` everywhere allows you to also customize the data directory location in the future, if needed. Refer to the API reference of `ApplicationData` and

`ApplicationDataOverride` for details

— how it is autodetected, and how to customize it. More information about the data directory is in our manual: https://castle-engine.io/manual_data_directory.php.

There is no other directory name that is "special" for Castle Game Engine. All other directory names you invent yourself, and organize your data and code however you like. Using `ApplicationData` is a good approach for typical games, that are distributed with a read-only game data. You do not have to use it. For example a general 3D model viewer or editor can just open 3D models from any user-specified URL (like a file or HTTP resource), not only from `ApplicationData`.

Examine Camera

If you click and drag around, you will notice that camera navigation already works. The engine created by default a camera that matches the position and size of your scene.

And by default it uses the "Examine" navigation type, which allows to comfortably drag and scroll to inspect the model. Many camera settings can be configured inside the model file assigned to the `SceneManager.MainScene`.

As it happens, our

`soldier1.castle-animframes`

doesn't set anything, but other model could.

For example, if your model in Blender has a Blender camera object then it will determine the default camera position and orientation.

Camera settings can also be adjusted by Pascal code. We will do this in the section "Walk Camera" later.

Play Animation

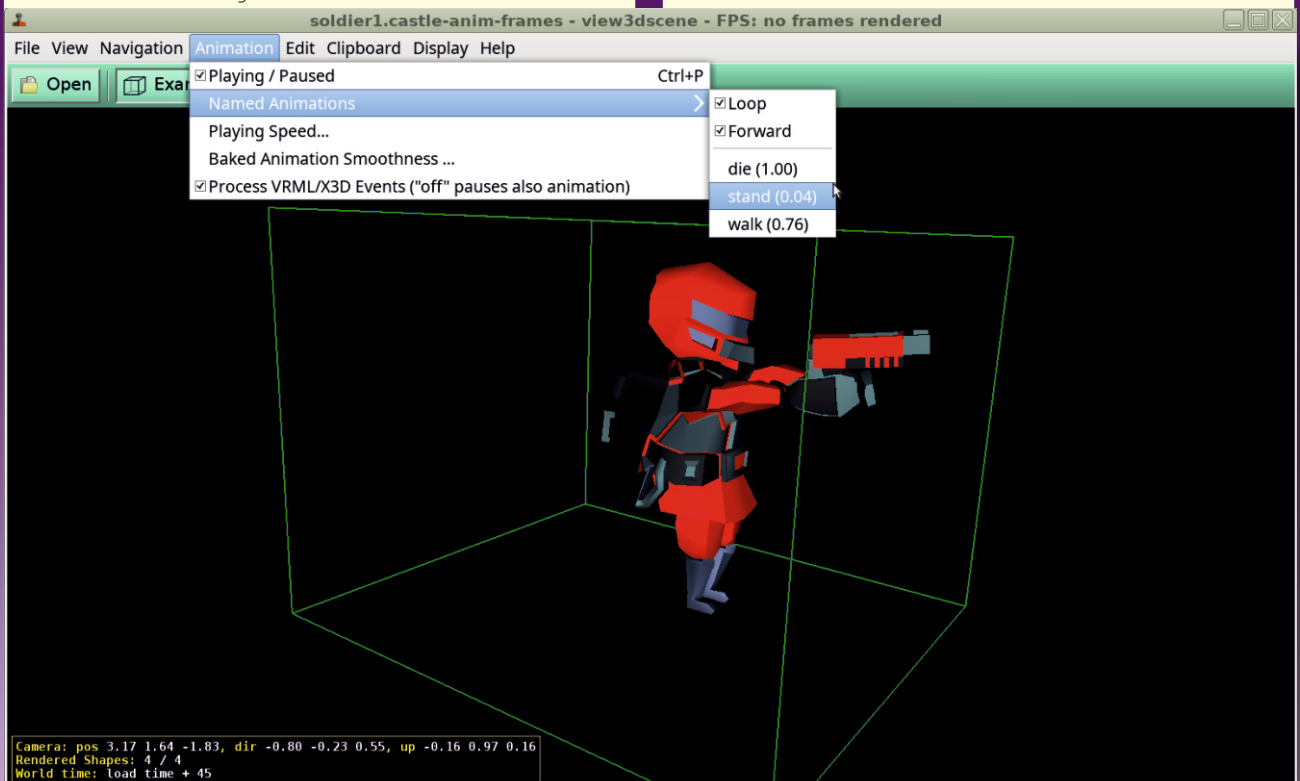
To play an animation, simply call the `TCastleScene.PlayAnimation` method with the name of your animation.

You also have to make sure that the scene processes events, which is a general mechanism responsible for any dynamic changes in a scene. To do this, set the `TCastleScene.ProcessEvents` property to true.

This is the code you can add at the end of `ApplicationInitialize`:

```
SoldierScene.ProcessEvents := true;
SoldierScene.PlayAnimation('walk', paForceLooping);
```

The 2nd parameter of `PlayAnimation` determines should the application repeat in a loop. Since **Castle Game Engine 6.5** there is also an overloaded version of `PlayAnimation` that allows to set more options through



`TPlayAnimationParameters` , and allows to play animation backwards, blend (cross-fade) the animation with the previously playing animation, get a notification when animation stops and more.

To know what animations are supported on a given model, it's easiest to open the model with our tool `view3dscene` . You can download it from <https://castleengine.io/view3dscene.php> .

You can see the available animations, and try them out, in `view3dscene` .

This information is of course also available through the Castle Game Engine API. Use `TCastleScene` methods `AnimationsList` , `HasAnimation` and `AnimationDuration` .

The animation names are set within your 3D modeling software. In case of exporting from Blender to `castle-anim-frames` format, the animation names simply correspond to Blender's action names.

React to User Input

Our `soldier1.castle-anim-frames` model has three animations: `die` , `stand` , `walk` . In the future, we will want these animations to be controlled through some game logic.

But for now, let's simply test whether they look good, by switching the current animation when we press keys 1 , 2 or 3 .

To react to user pressing something (a key, a mouse button, a touch on mobile device) it is easiest to assign a procedure to the `Window.OnPress` event. The event gets a parameter that says what key was pressed.

1. Somewhere in your `ApplicationInitialize` add this:
`Window.OnPress := @WindowPress;`
2. Define (above the `ApplicationInitialize`) a new procedure `WindowPress` , like this:

```
procedure WindowPress(Container: TUIContainer; const Event: TInputPressRelease);
begin
    if Event.IsKey('1') then SoldierScene.PlayAnimation('walk', paForceLooping)
    else
    if Event.IsKey('2') then SoldierScene.PlayAnimation('stand', paForceLooping)
    else
    if Event.IsKey('3') then SoldierScene.PlayAnimation('die', paForceLooping);
end;
```

3. Add `CastleKeysMouse` to the `uses` clause of your unit, in order to have the identifier `TInputPressRelease` available.

That's it. You can now switch the soldier's animation by pressing the appropriate key. Larger games typically have many states, like main menu, playing game and game paused. It would be uncomfortable to handle everything in one central `Window.OnPress` handler in such case. Instead, you can use the `TUIState` class. You can create multiple `TUIState` descendants, like `TMainMenuState` , `TPlayGameState` , `TGamePausedState` . Each descendant can override `TUIState.Press` to provide it's own input handling.

Conceptually, this is very similar to having multiple `TForm` instances in a regular Lazarus LCL or Delphi VCL application. Our `TUIState` has various special features, e.g. you can have a stack of states, so that a paused state is displayed on top of a (possibly frozen) game state.

In general, you can create custom `TUIControl` descendants, and check pressed keys within the `TUIControl.Press` overridden method.

The `TUIState` class is only a special `TUIControl` descendant. Our documentation https://castle-engine.io/manual_2d_user_interface.php describes `TUIControl` and `TUIState` in more details.

Load 3D Model of the Game Level

Let us add to the game world

(SceneManager.Items) another 3D model, representing a level. This is simply a matter of creating a new TCastleScene instance and adding it to the SceneManager.Items , alongside existing SoldierScene .

We also want the level scene to define lights that shine on all other scenes, including the soldier.

To do this, we change SceneManager.MainScene to point to the new LevelScene , not SoldierScene . The lights in MainScene automatically shine on all scenes (thanks to

SceneManager.UseGlobalLights).

In summary, this is the new code of our ApplicationInitialize :

```
procedure ApplicationInitialize;
begin
    Window.OnPress := @WindowPress;
    SoldierScene := TCastleScene.Create(Application);
    SoldierScene.Load(ApplicationData('character/soldier1.castle-animframes'));
    SoldierScene.ProcessEvents := true;
    SoldierScene.PlayAnimation('walk', paForceLooping);
    Window.SceneManager.Items.Add(SoldierScene);
    LevelScene := TCastleScene.Create(Application);
    LevelScene.Load(ApplicationData('level/level-dungeon.x3d'));
    Window.SceneManager.Items.Add(LevelScene);
    Window.SceneManager.MainScene := LevelScene;
end;
```

Be sure to also declare the variable LevelScene: TCastleScene somewhere above, alongside the SoldierScene . It could also be declared as a local variable

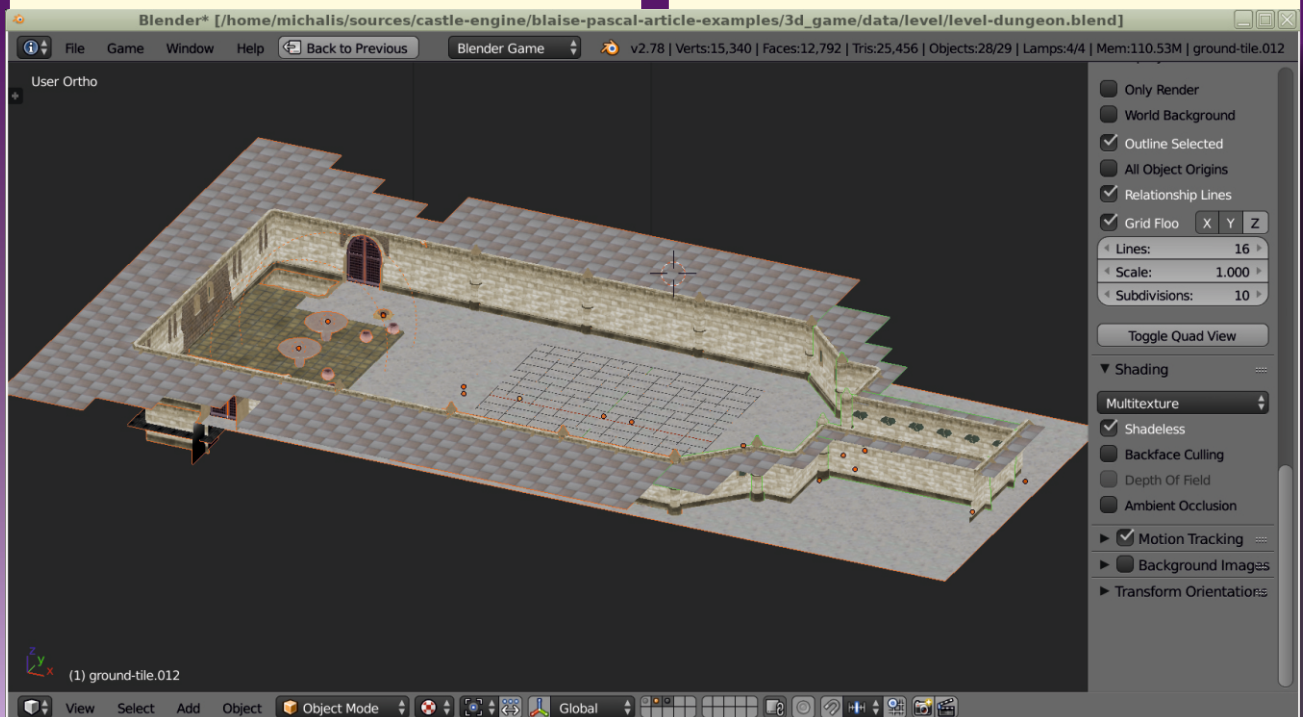
within ApplicationInitialize in this simple example. To make the lighting on our level be rendered with better quality, you can use Phong shading. Simply add this line somewhere at the end of ApplicationInitialize :

```
LevelScene.Attributes.PhongShading := true;
```

You can get a sample model of a level from our repository

<https://github.com/castleengine/blaise-pascal-article-examples/>.

The sample level is inside the 3d_game/data/level/ subdirectory. It is a Blender model (level-dungeon.blend) exported to an X3D (level-dungeon.x3d) format.



The level refers to a number of textures inside the textures/ subdirectory. The level-dungeon.x3d refers to the textures using a relative path, like `url="textures/wall-tex-1.jpg"` (simplifying a little).

This means that you can copy the entire model to your project, under any subdirectory you like, just make sure to keep the textures directory alongside the level-dungeon.x3d file. You can run the game now, and behold two 3D models displayed at once.

You can set camera position and orientation using the appropriate methods, like

`SceneManager.WalkCamera.Position` or `SceneManager.WalkCamera.SetView`.

For this demo, the ntWalk navigation will be a good starting point. The navigation mode constants are defined in the CastleCameras unit, so add it to your uses clause.



Walk Camera

The engine features a ready implementation of a walking or flying camera. It automatically responds to the typical keys used in 3D games, so you can move using arrow keys or AWSD key combination. The camera can also be affected by gravity, so you are pulled down in the negative Y axis (by default) until you stand on the ground. To switch to walk navigation, simply add this line to the `ApplicationInitialize` procedure:

```
Window.SceneManager.NavigationType := ntWalk;
```

There are other navigation types, like `ntFly`, `ntNone`, `ntExamine`. Using the None navigation type you can disable any built-in camera navigation, so that camera will stay completely stationary, regardless of pressed keys, gravity and collisions.

You are expected then to implement 100% of the camera navigation yourself.

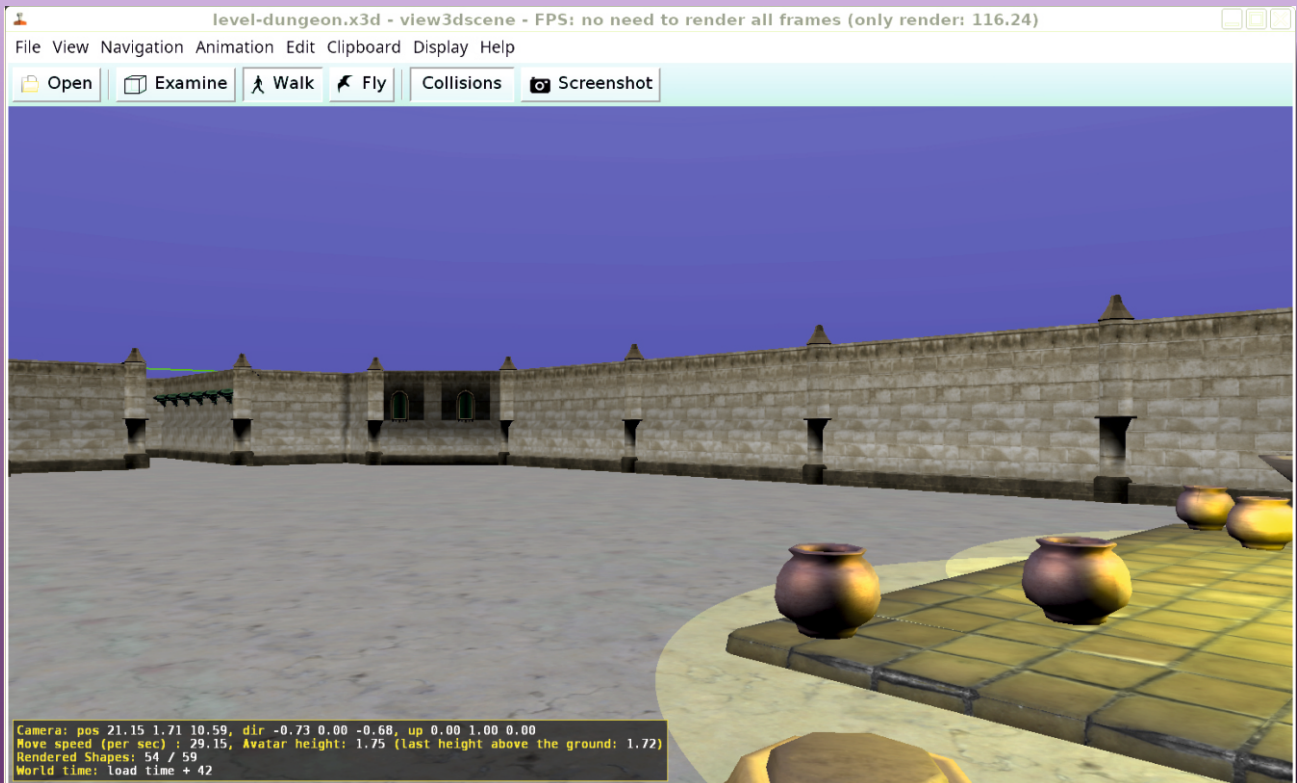
Let's make the camera move a little faster than default, by adding this command:

```
Window.SceneManager.WalkCamera.MoveSpeed := 10;
```

We should also set a good camera starting point. There are many ways to do this. For this demo, I advice using `view3dscene` to determine best camera position and orientation. **View3dscene** is a model browser developed using Castle Game Engine.

It can open anything that our engine can open, and it can be used to test engine rendering, animations and more. You can download it from <https://castle-engine.io/view3dscene.php>.

Open the level model in `view3dscene`, navigate to a proper place (you can use camera Examine or Walk modes interchangeably), and then look at `view3dscene` status bar that shows camera position, direction and up. Simply copy these values (you can use the menu item "Clipboard → Print Current Camera (Viewpoint) (Pascal)" since `view3dscene` 3.19.0). See the screenshot next page.



This is one possible camera starting point:

```

Window.SceneManager.WalkCamera.SetView(
Vector3(21.15, 1.71, 10.59), // position
Vector3(-0.73, 0.00, -0.68), // direction
Vector3(0.00, 1.00, 0.00),   // up (current)
Vector3(0.00, 1.00, 0.00)); // gravity up
  
```

The Vector3 function is available in the unit CastleVectors. Be sure to add it to your uses clause. In order for collisions with level to work precisely, you also need to add `ssDynamicCollisions` to the `LevelScene.Spatial` property. Otherwise, the LevelScene will collide as a one giant box, and you will not be able to move inside the level.

It is also a good idea to add `ssRendering` to `LevelScene.Spatial` property, while we're at it. This enables frustum culling, which is a good optimization when scene has many shapes, and often some of them are completely outside of the camera view (frustum). This is typically a useful optimization for game levels.

In summary, add this to the ApplicationInitialize :

```

LevelScene.Spatial :=
[ssRendering, ssDynamicCollisions];
  
```

You can now run the game, and freely walk using the AWSD and arrow keys.

Would you like to add mouse look, to enable rotating the camera by moving the mouse? Go for it! Handle in `WindowPress` some key and toggle the boolean property `Window.SceneManager.WalkCamera.MouseLook`. When it is `true`, the mouse cursor is invisible, and moving the mouse rotates the camera.

Enemy Intelligence

Right now, our enemy (the soldier model) just stands in place at the middle of the level. Let's implement a simple logic of a moving enemy. Let's also make it reusable, such that we will be able to spawn multiple soldiers in the future. To implement enemy logic we need to react to an update event of the engine. The engine makes sure that such event occurs regularly. In the typical circumstances, it just occurs as often as we render a frame.

There are various ways to handle this event, for example you can assign some procedure to `Window.OnUpdate`, or you can override the `TCastleTransform.Update` or `TUIControl.Update` methods.

For this demo, we will create a new class called `TEnemy` that descends from `TCastleTransform`. We will override `TCastleTransform.Update` method.

What is a TCastleTransform ?

- It is a group of 3D or 2D game objects. The children of TCastleTransform are added using the TCastleTransform.Add method, and they can be any other TCastleTransform instances. In particular, our familiar TCastleScene is a descendant of TCastleTransform, so it can also be a child of TCastleTransform.
- TCastleTransform can move (translate), rotate and scale the children.

In the case of our TEnemy, it will be a descendant of TCastleTransform, and it will contain exactly one child: a TCastleScene displaying the soldier. Other class designs are possible, for example we could implement TEnemy as a descendant of a TCastleScene. But I found the approach presented here most flexible. It allows to easily experiment and e.g. replace the visible portion of the enemy, or compose it from multiple scenes. In the overridden TEnemy.Update, we will change our own Translation value. The Translation is a 3D vector (TVector3 type). Notes about changing Translation in Update:

- Everything you do inside the Update should be scaled by a SecondsPassed parameter. SecondsPassed is the fraction of a second (as a floating-point value) that passed since the last Update call. For example, to move with a speed of 10 units per seconds, do not do this:

```
Translation := Translation + Vector3(10, 0, 0);
```

Instead do this:

```
Translation := Translation + Vector3(10 * SecondsPassed, 0, 0);
```

- By default, in Castle Game Engine, Y is the vertical axis, along which gravity works. So to make a horizontal movement, we move along the X and/or Z axis. This is consistent with default X3D, OpenGL, 2D conventions, and other game engines. Note that various 3D modeling software (like Blender) by default follow a different convention, in which Z is the vertical axis. However exporters (e.g. from Blender to X3D) will rotate your model to turn Z axis into Y axis.

This is the first version of our TEnemy class:

```
type
  TEnemy = class(TCastleTransform)
  public
    SoldierScene: TCastleScene;
    MoveDirection: Integer; //< Always 1 or -1
    constructor Create(AOwner: TComponent);
  override;
  procedure Update(const SecondsPassed: Single;
    var RemoveMe: TRemoveType); override;
  end;

  constructor TEnemy.Create(AOwner: TComponent);
  begin
    inherited;
    MoveDirection := -1;
    SoldierScene := TCastleScene.Create(Self);
    SoldierScene.Load(ApplicationData(
      'character/soldier1.castle-
        animframes'));
    SoldierScene.ProcessEvents := true;
    SoldierScene.PlayAnimation('walk',
      paForceLooping);

    Add(SoldierScene);
  end;

  procedure TEnemy.Update(const SecondsPassed:
    Single; var RemoveMe: TRemoveType);
  const
    MovingSpeed = 2;
  begin
    inherited;

    // We modify the Z coordinate,
    // responsible for enemy going forward
    Translation := Translation +
      Vector3(0, 0,
        MoveDirection * SecondsPassed * MovingSpeed);

    // Toggle MoveDirection between 1 and -1
    if Translation.Z > 5
    then MoveDirection := -1
    else
      if Translation.Z < -5 then MoveDirection := 1;
  end;
```


As you can see, we have moved the creation of the `SoldierScene` to the `TEnemy` constructor, and the `SoldierScene` is now a field inside each `TEnemy` instance. You should now remove the global `SoldierScene` variable and instead add `TEnemy` instance to the `SceneManager.Items` inside `ApplicationInitialize`, like this:

```
procedure ApplicationInitialize;
var
    Enemy: TEnemy;
begin
    // ...
    { These two lines should replace previous code
      dealing with SoldierScene.
      Keep the rest of ApplicationInitialize intact. }

    Enemy := TEnemy.Create(Application);
    Window.SceneManager.Items.Add(Enemy);
    // ...
end;
```

Note that it is also time to remove the debug code inside `WindowPress` that changes `SoldierScene` animation. In principle, we could add it now to overridden `TEnemy.Press`, but in the long run we want our game logic to dictate the animations of our enemy.

Be sure to add `CastleTransform` and `Classes` to the `uses` clause, to make the necessary types available.

There is one final tweak we should make: right now the enemy never rotates, so sometimes he walks backwards. To fix this, we should adjust the rotation of the transformed model.

One way to do this would be to change the `TCastleTransform.Rotation` property. It is a 4D vector, where the first 3 components describe the axis of rotation, and the 4th component is the rotation angle in radians.

However, rotating the creature using the `Rotation` property is not very comfortable.

It tells the engine how to rotate the loaded model, but we would prefer to tell the engine in which direction should the soldier look.

To do this, you can assign a `Direction` property, that (together with `Up`) actually modifies the `Rotation` under the hood.

Thanks to this, we can trivially enhance our `TEnemy.Update` implementation. Simply add there this line:

```
Direction := Vector3(0, 0, MoveDirection);
```

The enemy now rotates and always walks forward in our game. If you'd like, you can implement much more intelligent enemy that respects level's collision structure. The enemy can test the world around him using methods like `LineOfSight`, `Height`, `MoveAllowed`.

Spawning Multiple Enemies

Since we have created a nice reusable class `TEnemy`, it's only fair to use it to spawn multiple enemies. Do this by creating a couple of `TEnemy` instances.

Replace the previous code creating a single `Enemy` with the snippet below to create 10 enemies. Each enemy starts at a different `X` and `Z` coordinate.

```
for I := 0 to 9 do
begin
    Enemy := TEnemy.Create(Application);
    Enemy.Translation := Vector3(-5 + I * 1.5, 0,
                                RandomFloatRange(-5, 5));
    Window.SceneManager.Items.Add(Enemy);
end;
```

Add `CastleUtils` unit to the `uses` clause to have `RandomFloatRange` routine available.

You can try this code and it will work correctly. Multiple enemies walk in the world. However... the game loads surprisingly slow. The creation of 10 enemies takes a noticeable time.

You could use `ProcessTimer` from the `CastleTimeUtils` unit to actually measure the time spent within some part of the code, like the loop above.

The reason of the large loading time is that each `TEnemy.Create` loads the `soldier1.castle-anim-frames` file from disk, calling the `SoldierScene.Load(...)`.

Loading stuff from disk always takes a while, and here we do it 10 times when in fact we should load it from disk only once. As it happens, the `castle-anim-frames` file is also a bit large (animations in other formats would load much faster).

A more efficient solution is to create a single `TCastleScene` instance (let's call it `SoldierSceneTemplate`) and within each `TEnemy.Create` only call

```
SoldierScene := SoldierSceneTemplate.Clone
```

The `Clone` method creates a copy of the scene. The clone looks and behaves the same, but is independent from the original. Each scene clone may be in a different state (e.g. play a different animation). Do it like this:

1. Declare variable `SoldierSceneTemplate`: `TCastleScene` at the beginning of the unit implementation.

2. At the beginning of `ApplicationInitialize` (before the loop doing `TEnemy.Create`) initialize `SoldierSceneTemplate` like this:

```
SoldierSceneTemplate :=  
    TCastleScene.Create(Application);  
SoldierSceneTemplate.Load(ApplicationData(  
    'character/soldier1.castleanim-  
        frames'));
```

3. Change `TEnemy` constructor to use `SoldierSceneTemplate.Clone`:

```
constructor TEnemy.Create(AOwner: TComponent);  
begin  
  
    inherited;  
    MoveDirection := -1;  
  
    SoldierScene := SoldierSceneTemplate.Clone(Self);  
    SoldierScene.ProcessEvents := true;  
    SoldierScene.PlayAnimation('walk', paForceLooping);  
  
    Add(SoldierScene);  
end;
```

You should test the new version and see that it loads much faster.



In this simple example, we could also just add the `SoldierSceneTemplate` instance as a child of `TEnemy`, without even calling `SoldierSceneTemplate.Clone`. It is allowed to place the same `TCastleScene` multiple times in the world, and everything would work. However, all the enemies would then always have to play the same animation at the same time. This would prevent us from switching some enemies to die animation, which we will do in the next chapter.

Shooting at Enemies

This section shows how to implement (instantaneous) shooting or picking objects using Castle Game Engine.

The engine tracks the object under the mouse cursor, and exposes it under the `SceneManager.MouseRayHit` property. If `SceneManager.MouseRayHit` is `nil`, then we are not pointing the mouse at anything collidable. Otherwise `SceneManager.MouseRayHit` is a list of `TCastleTransform` instances that are hit by the ray. The first item on this list is the `TCastleScene` with which the collision occurred, and the following items are parent `TCastleTransform` instances of this scene.

In our case, we know that `TEnemy` instance is always a direct parent of a `TCastleScene` showing the enemy. Therefore we can check whether `SceneManager.MouseRayHit[1]` (a second object on this list) corresponds to a `TEnemy` class.

This is a new implementation of our `WindowPress` procedure that detects when we shoot the enemy.

When the enemy is hit, we do a couple of things:

1. Change enemy's animation to `die`.
2. Set a new boolean field `TEnemy.Dead` that you can use to stop moving the dead enemy inside `TEnemy.Update`.
3. Disable `Pickable` of `TEnemy`, to prevent it from being detected as shot again.
4. Disable `Collides` of `TEnemy`, to allow to easily walk over corpses of your enemies.

Note that this also works on mobile, and when we use mouse look. The

`SceneManager.MouseRayHit` always corresponds to the last thing hit by the pointer (mouse or touch). In case of mouse look, the cursor is always in the middle of the screen.

There are other places that contain detailed information about the picked object. `SceneManager.TriangleHit` describes the details of a triangle under the mouse. You can also call collision routines like `SceneManager.Items.WorldRay` to pick using any ray (not necessarily corresponding to the current mouse position).

Playing Sound and Music

At the end, we can add some sound and music to the game. It's so easy and fun that it would be a shame not to do this.

Castle Game Engine has a powerful audio support in the `CastleSoundEngine` unit. This unit exposes `SoundEngine` singleton which is a central place to initialize and play sounds.

```
procedure WindowPress(Container: TUIContainer; const Event: TInputPressRelease);
var HitEnemy: TEnemy;
begin
    if Event.IsMouseButton(mbLeft) then
    begin
        if (Window.SceneManager.MouseRayHit <> nil) and (Window.SceneManager.MouseRayHit.Count >= 2)
            and (Window.SceneManager.MouseRayHit[1].Item is TEnemy) then
        begin
            HitEnemy := Window.SceneManager.MouseRayHit[1].Item as TEnemy;
            HitEnemy.SoldierScene.PlayAnimation('die', paForceNotLooping);
            HitEnemy.SoldierScene.Pickable := false;
            HitEnemy.SoldierScene.Collides := false;
            HitEnemy.Dead := true;
        end;
    end;
end;
```

You can load sound files in .wav or .ogg (OggVorbis) formats. Sounds can be looping and you can tweak various parameters, like volume and pitch of each sound. Sounds can even be spatial, which means that they play in the proper channel (left, right or other) with the appropriate intensity, depending on their relative position to the player in 3D.

For starters, it is easiest to define a set of named sounds using a special XML file.

We have included some sample audio files in the <https://github.com/castle-engine/blaise-pascal-article-examples/> repository, in the subdirectory 3d_game/data/audio/. They were created by <https://opengameart.org/> contributors, their details are in the 3d_game/data/AUTHORS.txt file. The instructions below assume you have copied the dark_fallout.ogg and flaunch.wav files to your project, to the subdirectory data/audio/. Create the following text file, and save it as data/audio/index.xml within your project:

```
<?xml version="1.0" encoding="utf-8"?>
<sounds>
  <sound name="dark_music" url="dark_fallout.ogg"/>
  <sound name="shoot_sound" url="flaunch.wav"/>
</sounds>
```

Now you can initialize sounds inside ApplicationInitialize by loading this file:

```
SoundEngine.RepositoryURL :=
  ApplicationData('audio/index.xml');
```

To play music simply set

```
SoundEngine.MusicPlayer.Sound.
```

You can do this from ApplicationInitialize, right after the line above:

```
SoundEngine.MusicPlayer.Sound :=
  SoundEngine.SoundFromName('dark_music');
```

Finally, to play the shooting sound, call the SoundEngine.Sound method. You can place this inside the WindowPress procedure:

```
SoundEngine.Sound(SoundEngine.SoundFromName(
  'shoot_sound'));
```

This is it. You should now hear music and shooting sounds in your game.

You need to have the appropriate libraries installed to be able to play sounds and load OggVorbis music file.

- Under Linux, you should install appropriate packages.
- Under Windows, you should get the appropriate .dll files and place them alongside your .exe file. The necessary files are inside the Castle Game Engine zip that you already downloaded, look in the tools/build-tool/data/external_libraries/ subdirectory. See <https://castle-engine.io/documentation.php> for more details.

BUILDING AN ANDROID VERSION

Using the Build Tool to Compile for Android

To enable compiling the application with our build tool, create a file

CastleEngineManifest.xml in the main project directory.

The main purpose of this file is to specify the game_units attribute. It tells us which

unit is responsible for creating a window (Application.MainWindow). This is the GameInitialize unit in our case. When compiling for Android or iOS, we create a special library which will use the units mentioned in the game_units attribute. The window creation will be controlled by this library.

These are the contents of a simple CastleEngineManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<project name="my_game"
  standalone_source="my_game.lpr"
  game_units="GameInitialize">

  <!--
    Using "integrated" project type will automatically
    include sound libraries (OpenAL, OggVorbis).
  -->

  <android project_type="integrated"/>
</project>
```


Many more interesting things can be specified there, see the documentation on <https://github.com/castle-engine/castle-engine/wiki/CastleEngineManifest.xml-examples>.

Now you will need to get our build tool, which is an executable called **castle-engine**. For now, you simply have to compile it yourself. You can open the project `castle_game_engine/tools/build-tool/code/castle-engine.lpi` and compile it from Lazarus. The resulting application (`.exe` file on Windows) is compiled to `castle_game_engine/tools/build-tool/` directory.

Finally, adjust the environment variables.

- You want to set the environment variable `CASTLE_ENGINE_PATH` to point to the directory containing the `castle_game_engine`, like `c:/my_projects/castle_game_engine` or `/home/me/my_projects/castle_game_engine`. If you don't know how to set an environment variable, please search it on the Internet, there are step-by-step instructions available for every operating system.
- You also want to adjust your environment variable `PATH` such that it points to the location where the compiled build tool is. For example, to `c:/my_projects/castle_game_engine/tools/build-tool/` or `/home/me/my_projects/castle_game_engine/tools/build-tool/`. Again, if are unsure how to do this, please search the Internet.

In the next engine release, we will provide precompiled binaries of various tools (like `castle-engine` and `castle-editor`) that will make most of the above steps not necessary. Once you have done this, you should be able to open a console and execute this:

```
castle-engine --version
```

In response, it should show the **Castle Game Engine** version.

Now enter (`cd ...`) into the directory of your project. You can compile and run it for your current (desktop) system by executing these commands:

```
castle-engine compile
castle-engine run
```

To compile and run it on Android, install Android SDK and FPC able to cross-compile for Android (see the documentation on <https://github.com/castle-engine/castle-engine/wiki/Android>).

Then execute this:

```
castle-engine package --os=android --cpu=arm
castle-engine install --os=android --cpu=arm
castle-engine run --os=android --cpu=arm
```

This will compile and create (`package`) Android apk file, install it on your phone (connected through the USB cable) and run the application (displaying the log). See the <https://github.com/castle-engine/castle-engine/wiki/Build-Tool> for more details about using the build tool.



Cross-platform Logging

It's a good idea to initialize logging as early as possible. To do this, add these lines at the beginning of your unit initialization section:

```
ApplicationProperties.ApplicationName :=  
,my_game';  
InitializeLog;
```

Make sure to also add

`CastleApplicationProperties` and
`CastleLog` units to your uses clause.

The engine automatically logs some important events (like creation of the drawing context).

You can also use yourself routines like `WritelnLog` and `WritelnWarning` to send information to the log.

Conclusion

I hope that this article encouraged you to develop cool games using **Castle Game Engine**. If you get lost, please remember that we have a ton of documentation on our website

<https://castle-engine.io/> : manual, API reference, introduction to the modern Pascal language, and more. And we are a welcoming community! You can post on our Discord or forum (see <https://castle-engine.io/talk.php>) and we'll be happy to help.

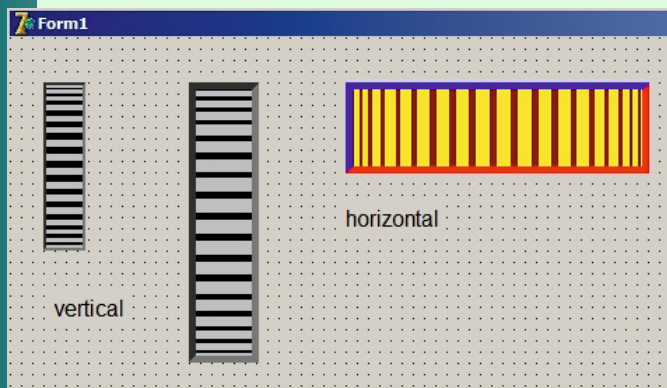
Consult also the examples code and data on <https://github.com/castle-engine/blaise-pascal-article-examples> .

Now, all you need to do is to take this variable and call it `PlayerHealth` . And another one, and it will be...



This article describes a rotating button component for the Delphi programming language. The button position is controlled by mouse movements.

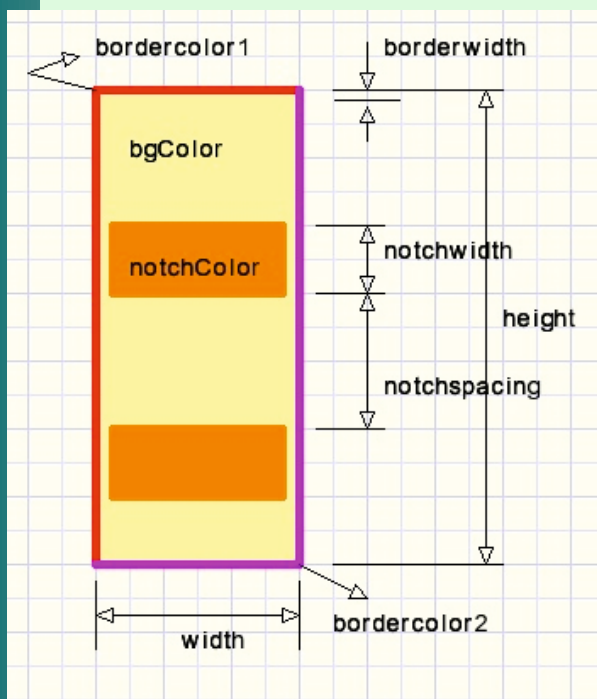
The application may be in the control of simulated laboratory equipment. The picture below shows some buttons on a form:



The buttons have a 3D effect showing rotation. A mousedown on a button followed by mouse movement changes the button position.

BUTTON PROPERTIES

The rotation button component is a descendant of the TGraphicControl class.



The picture at left below shows the (published) properties that shape a rotating button.

Painting is done on a bitmap, which is created when the button is painted for the first time. The bitmap is destroyed together with the button component. After drawing, the bitmap is copied to the button canvas. This prevents flickering which would result when painting was done directly on the button canvas.

The output position of the button is a number from 0 to 255 (byte). Because a pixel distance is very small it is not convenient to obtain the button position directly from the pixel where to mouse is pointing to. The following (published) properties control the button position:

- pixelratio : the number of pixels that increment or decrement the button position
- maximum : the maximum button position
- position : the current button position

ORIENTATION

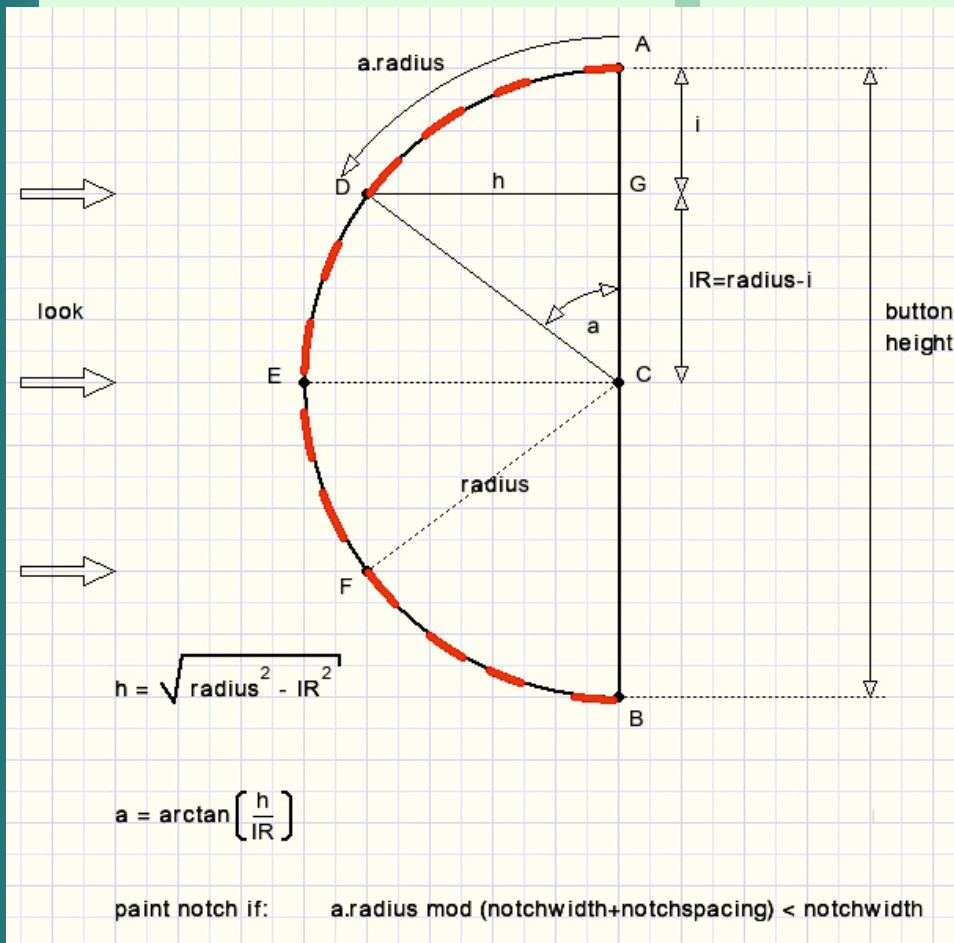
- orHorizontal : button changed by mouse movement in horizontal direction
- orVertical : button changed by mouse movement in vertical direction

EVENTS

- onChange
- onButtonPaint
- onEnter
- onLeave
- onChange
provides the new button position.
- onButtonPaint
This event is raised after painting of the bitmap is finished but before the bitmap is copied to the button canvas. So, this event enables modifications to the bitmap such as the display of values. Read property map to get the bitmap.
- onEnter, onLeave
These events occur when the mouse pointer enters or leaves the button.

PROGRAM DESCRIPTION OBTAINING THE 3D EFFECT

Consider a vertically oriented button:



Variable *i* steps from 0 at point A to the button height at point B. For each step the angle *a* is calculated (in radians). At center *C*, $a = p/2$. The $\arctan()$ function outputs negative values for quadrants 2 and 4. When passing C downwards (quadrant 2), *a* has to be incremented by *p*.

When moving the mouse over the button with the mousebutton pressed mousemove events provide the (x,y) coordinates of the mouse pointer. These (x,y) values are compared with previous values and the difference is the number of passed pixels. Variable pixelcount holds this button movement in pixels.

The button position simply is this pixelcount value divided by property pixelratio.

BUTTON ROTATION

Before I described the 3D effect, but there is no button rotation, the notches cannot move. Movement is done by adding the pixelcount to the circle arc (*a.radius*) before calculating the modulus.

Care must be taken to have the notches move in the mouse direction.

There is a tricky difference between vertical and horizontal oriented buttons. Please look at the source code for details.

THE DELPHI PROJECT

This project consists of

- form1, unit1 : buttons and code to test the component
- unit dav7rotationbtn which holds the TDav7RotationBtn class.

LAZARUS PROFESSIONAL



CONFERENCE

KÖLN/BONN

German and English Spoken

THURSDAY 20

FRIDAY 21

SATURDAY 22

SEPTEMBER 2018

LAZARUS PROFESSIONAL
CONFERENCE



KÖLN/BONN

IBEXPERT AND BLAISE PASCAL MAGAZINE

will organize in cooperation with the Lazarus Foundation and Lazarsu Factory

LAZARUS PROFESSIONAL CONFERENCE:

September 20 (Thursday) 21(Friday) 22 (Saturday) 2018.

ADDRESS:

Gustav-Stresemann-Institut e.V., Langer Grabenweg 68, DE-53175 Bonn

LOCATION AND APPROACH:

Langer Grabenweg 68 D-53175 Bonn

BY RAIL:

Between **Bonn central station** and Bonn-Bad Godesberg trams commute every 7 minutes (tram-number 16 and 63)

From **Bonn central-station**: U-Bahn Line (underground/tram number) 16 or 63, direction Bad Godesberg, leave tram at "Max-Löbner-Straße" - walk down "Max-Löbner-Strasse" to the end. From **ICE-Station** Siegburg / Bonn: U/tram-line 66, direction Bonn/Bad Honnef - leave tram at station "Robert-Schuman-Platz" Kurt-Georg-Kiesinger-Allee, turn left to Jean-Monet-Straße, turn left to Heinemann-Straße

BY PLANE:

From airport Cologne / Bonn: Bus No. SB 60 until Hauptbahnhof (Bonn central station)

From Bonn central-station: take U-Bahn Line (underground/tram number) 16 or 63, direction Bad Godesberg. Leave at "Max-Löbner-Strasse" - Walk down "Max-Löbner-Strasse" to the end

ALLES Vorträge werden auf Deutsch und Englischer Sprache ausgetragen

Workshop Themen 20.-21.09.2018

Donnerstag 20 September 2018

Der Lazarus Desktop

Die wichtigsten Projekteinstellungen

Konvertierung von Delphi Quelltexten

Konvertierung Tips und Tricks

Sprachunterschiede Delphi Lazarus fpc

Mittagessen für alle Teilnehmer

Online Package Manager

Komponenten für Lazarus

Report Lösungen

Online Ressourcen:

Die wichtigsten Tools, Foren und Webseiten

Debugger und Laufzeitmessungen

Abendessen für alle Teilnehmer

„Beer and Best Practice Unconference“

Die Teilnehmer entscheiden gemeinsam, wer welches Thema vortragen soll

Freibier und freie Softwaredrinks inklusive

Der Tagungsraum steht bis Mitternacht zur Verfügung

German and English Spoken <https://www.blaisepascalmagazine.eu/events/>

Freitag 20 September 2018

Datenbankzugriff mit SQLDB und Alternativen
Installation und erste Schritte unter Windows, Linux und Mac
Installation pas2js und erste Schritte zur Mobilen Anwendung auf iOS
und Android Distribution von pas2js Anwendungen
Lazarus und Firebird Datenbanken

Mittagessen für alle Teilnehmer

Softwarearchitektur für kundenspezifische Anwendungen
Business Logik, Multitier, Datenbank Prozeduren
Chromium Integration in Fat Client Anwendungen
TMS Webcore für Lazarus

„Beer and Best View“ Die Teilnehmer gehen gemeinsam zu einem Biergarten an den Rheinwiesen, sofern es das Wetter erlaubt. Abendessen und Getränke auf eigene Rechnung

Interessierte und Teilnehmer am Community Day sind herzlich eingeladen

Open End

German and English Spoken <https://www.blaisepascalmagazine.eu/events/>
Samstag Community Day 22.09.2018

Lazarus und fpc im Überblick: Roadmap
Der Umstieg von Delphi auf Lazarus
Komponenten: Fluch oder Segen

Mittagessen für alle Teilnehmer

Echte Multiplattform Anwendungen und Besonderheiten
Windows / Linux / Mac
Pas2js iOS/Android
Offene Diskussion: Was fehlt euch noch in Lazarus (oder was kennt ihr nur noch nicht)
Aushang: Mitarbeiter gesucht, Firmen suchen Pascal Entwickler, Pascal Entwickler suchen Arbeitgeber

Im Ticketpreis sind Getränke, Kaffee und ein Mittagessen, jedoch keine Übernachtungskosten enthalten. Die meisten Sessions werden in deutscher Sprache durchgeführt, wir übersetzen natürlich auch auf English und Holländisch da wir genug Mitarbeiter in diesen sprachen haben.



Anmeldung hier:

<https://www.lazprocon.net/anmeldung>

CONVERTING DELPHI CODE TO LAZARUS PAGE 1/8

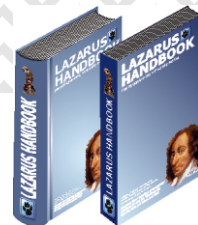
EXCERPT FROM THE LAZARUS HANDBOOK

BY JUHA MANNINEN AND HOWARD PAGE-CLARK

starter expert



At the source file level a project's `.lpr` is conceptually equivalent to a `.dpr`, and a `.lfm` is equivalent to a `.dfm`, and `.pas` files can be identical.



However, the Lazarus `.lpi` file has no exact equivalent in Delphi, and `.lpk` and `.dpk` files are completely incompatible.

In Lazarus a project's settings are stored in the project's `.lpi` and session file (`.lps`); whereas in Delphi various different project files such as `.dproj`, and `.groupproj` are used, and the number and names of the project support files is very Delphi-version-dependent.

Where Delphi code is now required to run under Lazarus there are three typical situations:

- You want to maintain the same codebase in both Delphi and Lazarus. This is certainly possible, within certain restrictions, and with judicious use of `{IFDEF ...}`. The Lazarus Delphi conversion engine may help if you are starting from Delphi code which you want to be compilable using FPC, rather than starting from scratch. The **IDE Converter** is not designed principally for this scenario, but it does offer a Use the same DFM form file option and a Support Delphi option in the Target section of the Convert dialog, which go some way towards this goal (see the Conversion Options section below).
- You want to convert a legacy Delphi Windows VCL project so you can maintain it using Lazarus. The project will continue to be Windows-only.
- You want to convert a Delphi project to be maintained as a fully cross-platform Lazarus project.

The converter provided by the Lazarus IDE is designed primarily to help with the conversion process in the latter two cases, relieving you of much of the drudgery the conversion process otherwise involves, perhaps leaving only minor issues to be resolved "by hand".

In this article we show a chapter of the Lazarus Handbook we are writing at this moment. We make this available because lots of our readers have been asking for this - especially since the enormous progress Lazarus and FPC have made. This actually is a very good help to plan for converting your project to Lazarus. There are some quite interesting paragraphs that shed also light on the differences between the two IDE's.

THE NEED FOR CONVERSION

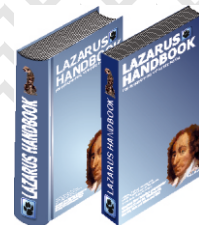
Delphi compatibility has been an important goal for Lazarus from the time it began. Many people who use **FPC/Lazarus** also work with **Delphi**, or are **ex-Delphi** users.

However **Lazarus** has been cross-platform from the outset, and for decades **Delphi** was Windows-only (if you discount the quickly abandoned **Kylix** experiment); and though **Delphi** now cross-compiles to several non-Windows platforms, it remains a Windows-only IDE.

While **FPC** and **Lazarus** strive to be as **Delphi-compatible** as possible, neither the respective compilers, nor the IDEs, nor the **Object Pascal** dialects, nor the default runtime libraries of the two development apps are identical.

Consequently it is not possible to simply open a Delphi project in Lazarus and expect that it will compile and run.





The converter automates certain adjustments and code replacements that almost any Delphi-to-Lazarus conversion will require. Because the Converter is designed for the Delphi → Lazarus direction only, it is less helpful with the first scenario outlined above, which might require a degree of Lazarus → Delphi conversion.

CONVERSION ISSUES AND LIMITATIONS

The **Lazarus Delphi Converter** is not designed to cope comprehensively with any possible Delphi code you might give it.

It is most successful at converting code which avoids third party components and uses only **standard VCL components**, but will assist you whatever your conversion needs might be. How much manual conversion work remains to be done following the automated conversion will be related to issues such as the following, all of which will require further code customisation before **FPC/Lazarus** will be able to successfully compile and run the converted file(s).

Some of the following issues may altogether preclude a successful fully cross-platform conversion:

- **the presence of Windows-specific features:**
Windows API calls, OLE or COM automation
- **the presence of i386 assembly**
(or other very low-level) code
- **dependence on third party components**
such as specialised grid or data-aware controls
- **use of Delphi language features not currently supported by FPC,**
such as anonymous methods or use of non-ASCII variable names
- **dependence on Delphi-specific features** such
as FireMonkey or DataSnap
- **custom painting** done outside of OS Paint messages, and similar Windows-isms
- **API differences between libraries**
used both in Delphi and Lazarus, such as those commonly used for reading and writing XML or image files.

THE LAZARUS DELPHI CONVERTER

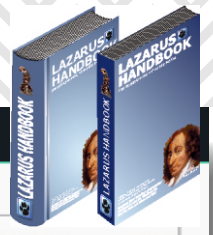
The Lazarus Delphi converter can convert an entire project or package, or merely a single unit or `.dfm` file. You access it via the Tools → Delphi Conversion menu, which offers four submenu options:

- **Convert Delphi Unit to Lazarus Unit...**
- **Convert Delphi Project to Lazarus Project...**
- **Convert Delphi Package to Lazarus Package...**
- **Convert Binary DFM to Text LFM + Check Syntax...**

You simply choose the option appropriate to your purpose. The last option is rarely needed because when converting a unit the `.dfm` form file is always automatically converted.

You are directed to locate the specific **Delphi unit/project/package/DFM** to convert, and presented with a settings dialog in which you specify various options which will govern the automated conversion process (see Figure 1 on the next page), which you initiate by clicking the Start Conversion button.

For simple one- or two-form Delphi VCL projects you may be able to accept all the default settings and simply press Start Conversion to end up with an **FPC-compileable Lazarus** project.



Convert Delphi project - Cannonballs3.dpr

Path:
/usr/SW/DelphiConversion/CannonBalls/Cannonballs...

Target

☒ Cross-platform

☐ Support Delphi

☐ Use the same DFM form file

Other

☒ Add defines simulating Delphi7

☐ Make backup of changed files

☒ Keep converted files open in editor

☒ Scan files in parent directory

Unit Replacements

Edit Automatic

Unknown properties

Automatic

Type Replacements

Edit Automatic

Function Replacements

Edit Enabled

☒ Add comment after replacement

Coordinate offsets

Edit Disabled

Help Cancel Start Conversion

Figure 1: The Convert Delphi project dialog preparing to convert a Delphi project available at delphiforfun.org

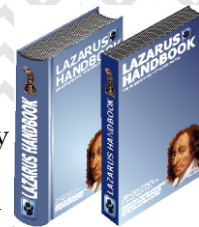
When you start the conversion process by clicking the Start Conversion button, your current dialog options are saved in the `delphiconverter.xml` file in your local Lazarus configuration directory. The dialog's Path field (read-only) shows the path of the `unit/project/package/.dfm` you selected for conversion.

CONVERSION OPTIONS

The Converter dialog offers two sets of options for customising the details of the conversion process. On the left are seven checkboxes which turn particular conversion functionalities on or off. On the right are five combo-boxes, which enable or disable automatic conversion features, in some cases allowing you to interact with the conversion process rather than letting it run completely automatically. Beside the dropdown combo-boxes is an Edit button which opens a further dialog where you can configure various specifics of the Converter's replacement routines.

Checkbox options

- Target
 - Cross-platform (checkbox default: True)
If `True`, this option replaces Windows-specific units (`Windows`, `WinProcs`, `WinTypes`) with their cross-platform equivalents (`LCLIntf`, `LCLType`, `LMessages`). This makes the LCL port of essential parts of the WinAPI available.
 - Support Delphi (checkbox default: False). If `False`, the original `.DFMs` are deleted, and only newly converted `LFMs` are used. If `True` new `.LFMs` are generated, and the original `.DFMs` are retained. To make units compatible with both kinds of form file, appropriate `{ $IFDEF... }` statements are inserted in each form's `.pas` file.
 - Use the same DFM file (checkbox default: False). With the default `False` setting, all Delphi `.DFMs` are converted to `.LFMs`. If `Support Delphi` is `False`, all `DFMs` are then deleted.



If `Use the same DFM file is True`, this setting does not generate any new `.LFMs`, but allows all original **Delphi** `.DFM` form files to be used with Lazarus (any binary `.DFMs` are converted to text-based `.DFMs`).

However this option is recommended only for very simple GUI forms.

It is problematic because VCL and LCL components have slightly different published properties.

Future maintenance is plagued by each IDE complaining about unknown properties, and adding properties unknown to the other IDE.

- **Other**

- **Add defines simulating Delphi7**
(checkbox default: `True`)

If `True` the following FPC compiler defines are added, which causes some projects to work better:

```
-dBorland -dVer150 -dDelphi7  
-dCompiler6_Up -dPUREPASCAL.
```

- **Make backup of changed files**
(checkbox default: `True`)

If `True` the converter first saves all original Delphi files to an auto-generated **ConverterBackup** directory under the selected directory. This is in addition to the usual Lazarus Backup subdirectory that is also created for Lazarus files.

Note that this does not work well if the code is converted in many steps.

The converter can work on partly converted Lazarus units, projects and packages, but in this case the earlier `ConverterBackup` is overwritten.

It is safer to make a backup of the original Delphi sources manually elsewhere if the conversion is likely to proceed in several stages.

- **Keep converted files open in editor**
(checkbox default: `False`)

If `True`, all converted unit files remain open after conversion. If `False`, only the converted project's (or package's) main file remains open in the editor.

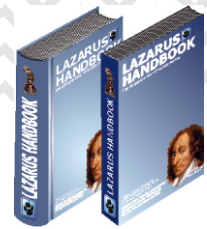
- **Scan files in parent directory**
(checkbox default: `True`)
Some Delphi projects have a directory structure in which the project directory is not the root of the structure. Unit files may then be found in sibling directories to the project directory. When this option is `True`, such directories are scanned for source files, which happens in a background thread, so the settings dialog remains responsive and the actual conversion proceeds unhindered when the **Start Conversion** button is pressed. Sometimes the parent directory has lots of unrelated code and this option can be disabled. Disabling takes effect only when the converter is next started.

DROPDOWN OPTIONS

- **Unit Replacements** (dropdown options: `Disabled`, `Interactive`, `Automatic`)
You can configure the unit replacements you want the Converter to make by clicking the **Edit** button.
This opens the **Units to replace** dialog, a two-column editor listing commonly encountered Delphi units in the left **Delphi Name** column, with their designated replacement unit listed on the right in the **New Name** column. As the Converter parses each Delphi unit's `uses` clause, all units listed in the **Units to replace** dialog are either removed or replaced, if they are found.
A blank New Name means the used unit will be removed. A non-blank entry means that unit will be replaced with the new unit name(s).
For example, by default `MMSystem` is removed since its **New Name** entry is blank, whereas `Windows` is replaced with its **New Name** entry:
`LCLIntf`, `LCLType`, `LMessages`.
Regular expression syntax for replacements using numbered parameters (`$1`, `$2` ...) is supported. For example:

* `^Q(.`+) in the **Delphi Name** column with `$1` in the **New Name** column means "remove a leading `Q` from the unit name", thus removing "`Q`" from old Kylix unit names.





- **Unit Replacements (continuing)**

The dropdown lets you specify whether unit replacement is Disabled, Interactive or Automatic. When interactive, the user can edit the replacements before they are applied. If units are still missing after the replacements, the converter asks you what to do about the missing units. You have the choice of commenting out the offending name(s), searching for them yourself, or abandoning the conversion at the stage reached so far, perhaps downloading missing units before trying again (see Figure 2).

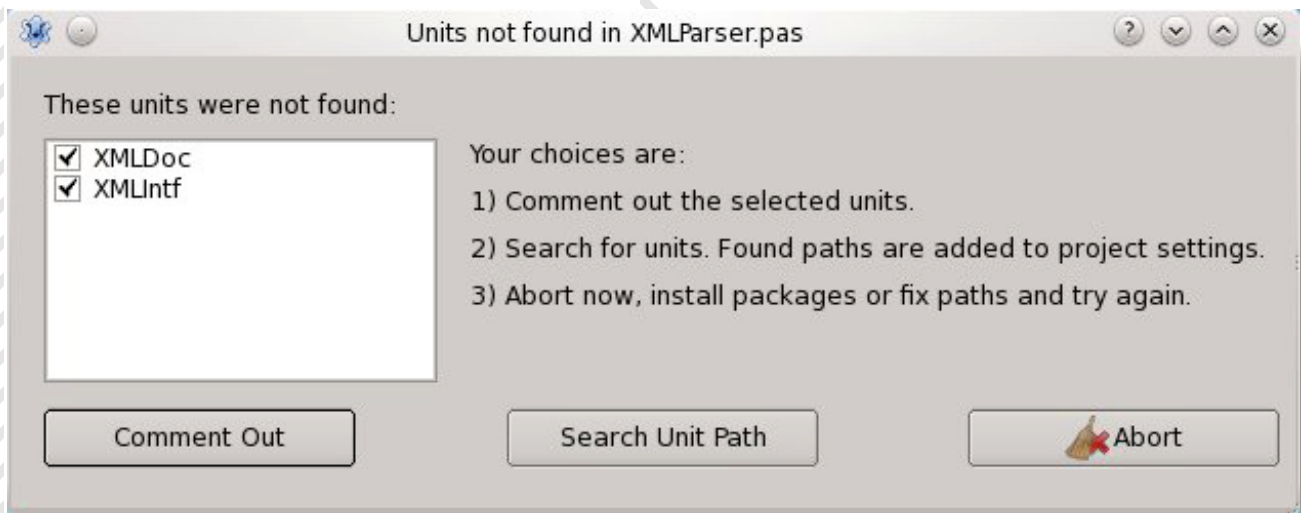


Figure 2: Warnig about units not found, so you can try to find them again

Unknown properties (dropdown options:

Disabled, Interactive, Automatic)

Some Delphi components' published properties do not have corresponding properties in their LCL counterpart components. Although Lazarus can cope with non-existent properties by registering them as such and so getting the form loader to ignore them, it is better to remove the non-existent properties. You can choose here whether the converter removes them automatically (*Automatic*) or whether the user will deal with their removal or replacement interactively (*Interactive*).

The interactive dialog gives scope for replacing the missing property, but that is rarely possible or necessary (see Figure 3).

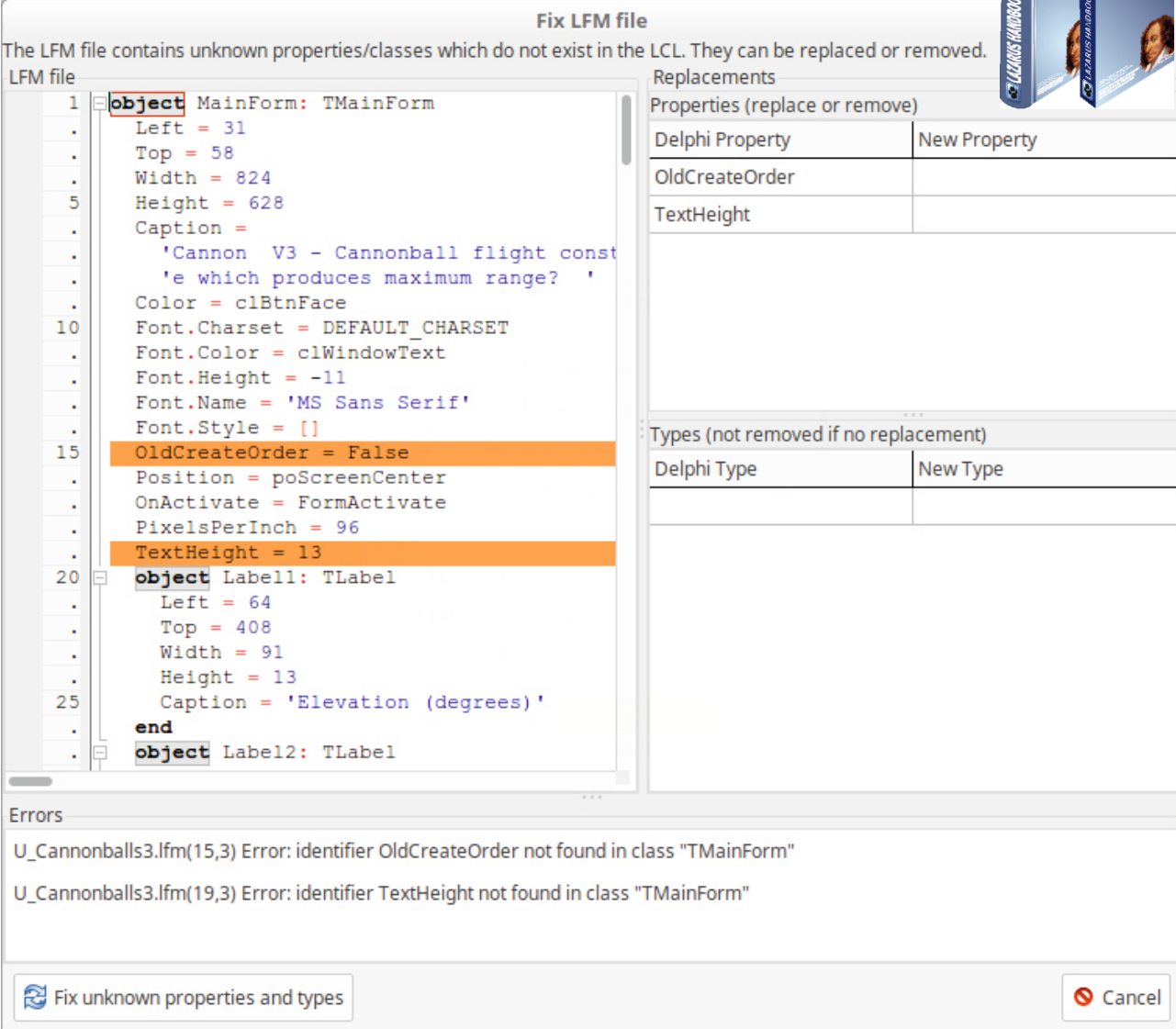
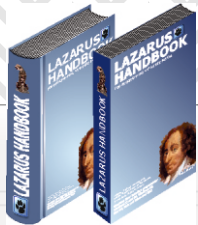


Figure 3: Unknown Properties in Interactive mode, showing a “Fix LFM file” dialog listing two non-existent LCL properties

- **Type replacements**
(dropdown options: Interactive, Automatic) Some Delphi components and many third party components have no corresponding LCL component of the same name. The Converter will replace many of those missing classes with fall-back LCL classes. For example: Delphi’s **TTabbedNotebook** can be replaced with the LCL **TPageControl**, and Delphi’s **TADOQuery** with FCL’s **TSQLQuery**. To configure the specifics of which LCL/FCL class replaces which Delphi class, press the Edit button to open the Types to replace

editor, which works just like the Units to replace dialog, accepting the same regular expression syntax (see above). Any listed component found is replaced in both the Pascal source file and in the form file (.lfm). Replacement even works for nested component structures, where a component to be replaced has child components that must also be replaced. Needless to say, where the replacement component has different properties from the original Delphi component you will encounter problems after conversion, for which there is no simple, automated solution.

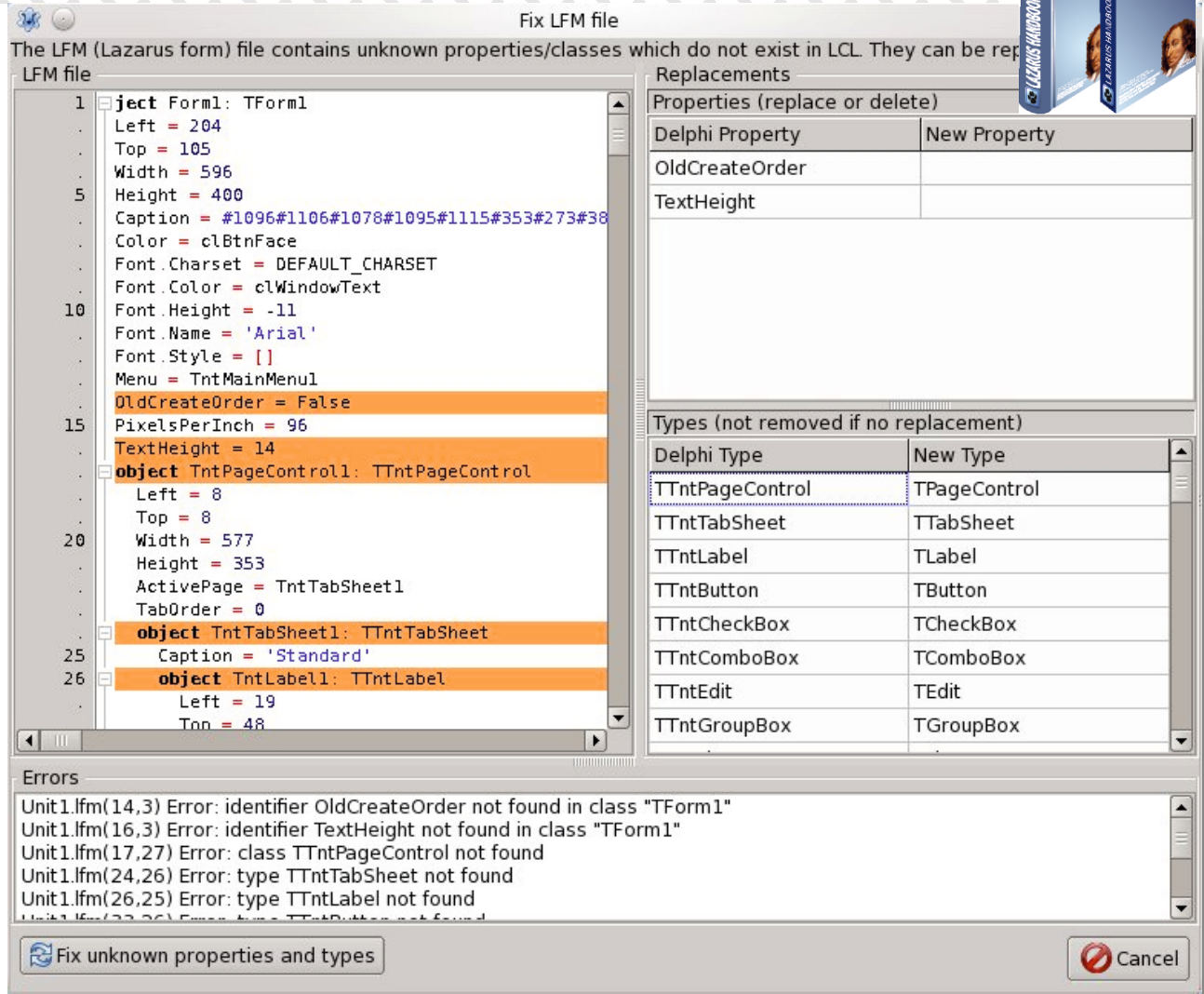
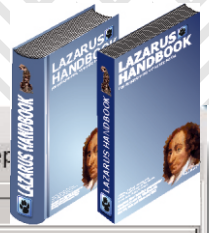


Figure 4: Interactive Type replacements dialog, listing several TTntxxx components with suggested LCL replacements

• Function Replacements:

(dropdown options: Disabled, Enabled)
Various Windows-only function calls in the Delphi source will need to be replaced with functionally similar RTL or LCL library function calls.

Usually the default replacements listed in the Functions/Procedures to replace dialog suffice. Pressing the Edit button opens this dialog if you need to change or add to the stock replacements already listed there. The functions are categorised, so you can enable or disable entire categories with a single click. The syntax uses \$1, \$2 etc. for successive parameters in the Delphi function call that is being replaced.

Thus the Delphi SameStr function is replaced by (CompareStr(\$1, \$2) = 0). A simple conditional

```
if paramNo match regularExpression then option1;
option2
```

syntax can be used for defining the replacement function. For example

```
ShellExecute → if $3 match "://"
then OpenURL($3); OpenDocument($3)
```

This allows ShellExecute to map to two different LCL functions. The string after match is a regular expression. If it matches, the then replacement is used,



otherwise the second replacement option following the semicolon is used. Since " : / " is typically found in URLs, the first match opens a URL, otherwise a local document is opened.

Add comment after replacement (checkbox)
A comment is automatically inserted by the converter after each replacement if this checkbox is checked. Usually it is helpful to see the replacements documented in your sources. You can then quickly identify where the code is that needs to be tweaked.

- **Coordinate offsets**

(dropdown options: Disabled, Enabled)

This option applies particularly to `TPanel`, `TGroupBox`, `TRadioGroup` and `TCheckGroup`. In Delphi the coordinates of contained controls are relative to their groupbox's `Top` and `Left`. In Lazarus the coordinates of contained controls are relative to their groupbox's client area – which ignores the border. This causes a problem for a converted `TGroupBox` because it has a title text on its top border. Most `TPanel` instances also have a double bevel border. Contained controls' coordinates therefore need to be adjusted by an offset to preserve the same form layout as seen in Delphi. Should you need to change the default offsets applied, press the Edit button to open a settings dialog where the needed adjustments are defined.

GLOBAL CONVERSION CHANGES

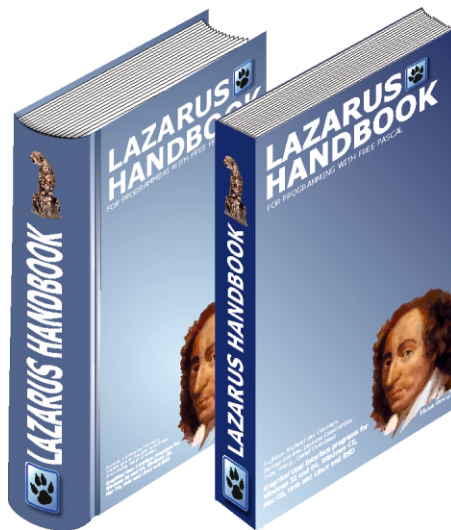
Depending on options you selected in the main Convert dialog, the following changes are applied to converted **Pascal** source and form files:

- a `{ $Mode Delphi }` directive is inserted in each source file, so the compiler will expect language syntax in **Delphi's Object Pascal** dialect. If you wish to use the features of the `{ $Mode objfpc }` dialect (or some other mode) you will need to alter the automatically inserted `{ $Mode Delphi }` directive later.
- File names of units in the uses clauses, and any include `(.inc)` file names are changed to match actual file names as used in case sensitive file systems.

- Various unit names in uses clauses are replaced or removed, and some Delphi function names and component class names are changed, depending on the settings in the Convert dialog.
- Any binary `.DFM` files are first converted to ASCII text format. `.DFM` files are replaced with `.LFM` files, or removed, or conditional compilation defines added according to your chosen Support Delphi setting in the Convert dialog. Properties unknown in Lazarus are normally removed, and Top and Left properties of contained controls are changed (depending on your chosen Unknown properties and Coordinate offsets setting in the Convert dialog).

Online information (which may be more recent than this documentation) can be found here:

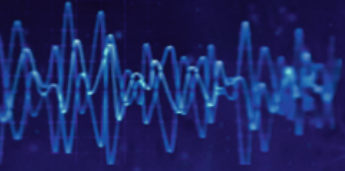
http://wiki.lazarus.freepascal.org/Delphi_Converter_in_Lazarus



<https://www.blaise Pascalmagazine.eu/blog/update-about-the-lazarus-handbook/>

MITOV SOFTWARE

WWW.MITOV.COM



Signal
Processing



Arduino



Audio

Computer
Vision



Video



Communication



Animation



AI

Visual
Instruments



Delphi
Components
Galaxy

Process Control

Mouse Click Away



starter expert

The way that these libraries are demonstrated, they are fully free and so easy to use that we believe even a very early starter can do this!

In the previous articles, I showed you how easy it is to play video files, how to filter and morph the video, apply effects, draw on frames, perform animations, and render on variety of 2D and 3D surfaces.

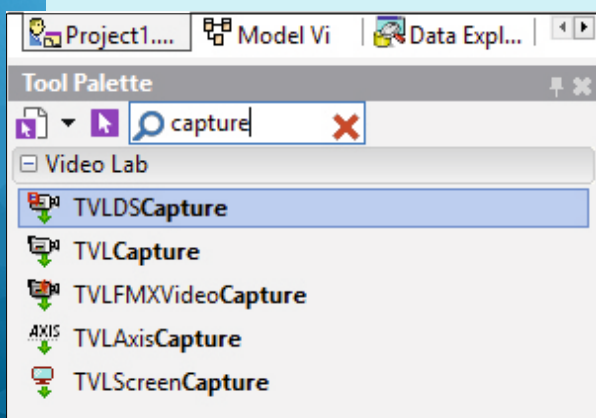
In all of the articles we used file player as a source for the video. VideoLab however includes a number of video source components - Video Players using variety of multimedia API's, Video Camera Capture components, Screen Capture, IP Camera and Web Stream receivers, local network stream receivers, and even components allowing you to generate your own video from frames.

In this article I will show you how easy it is to capture video from Video Cameras, TV Tuners, Remote IP Cameras, Internet Video Streams, and Images, and how to generate your own video from code.

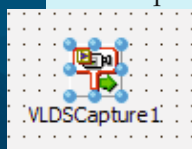
I will also show you how you can record the captured video, or broadcast it over the internet or to other computers on your network.

VIDEO CAPTURE

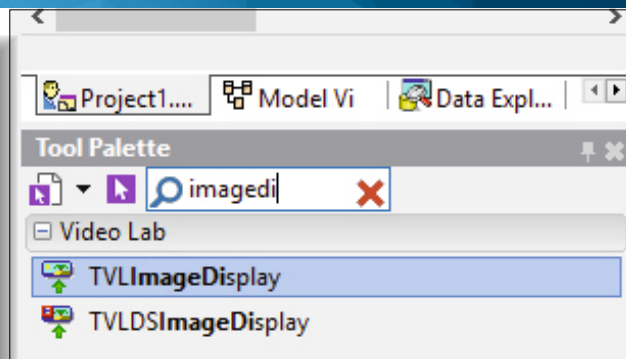
First we will create a video capture application using DirectShow video capture component. Start a new VCL Form application. Type "capture" in the Tool Palette search box, then select TVLDSCapture component from the palette:



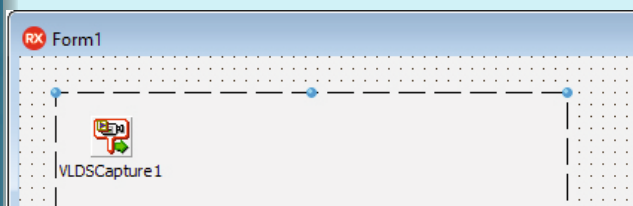
And drop it on the form.



Type "imagedi" in the **Tool Palette** search box, then select **TVLImageDisplay** from the palette:

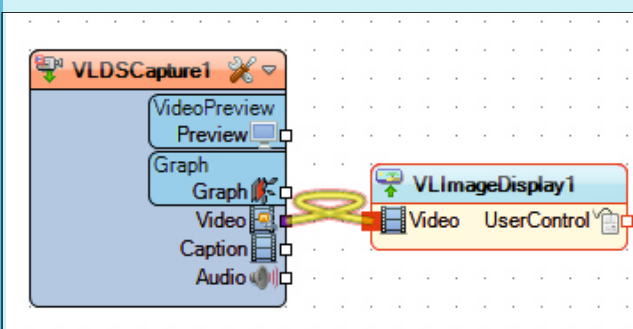



And drop it on the form:

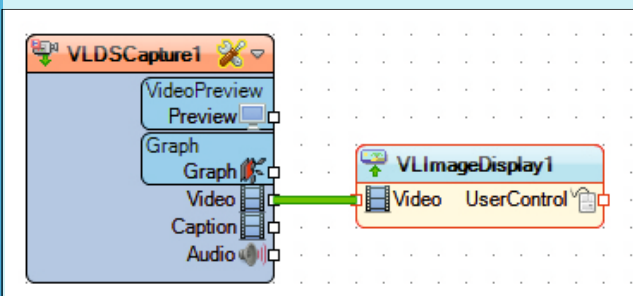


Switch to the "OpenWire" tab.

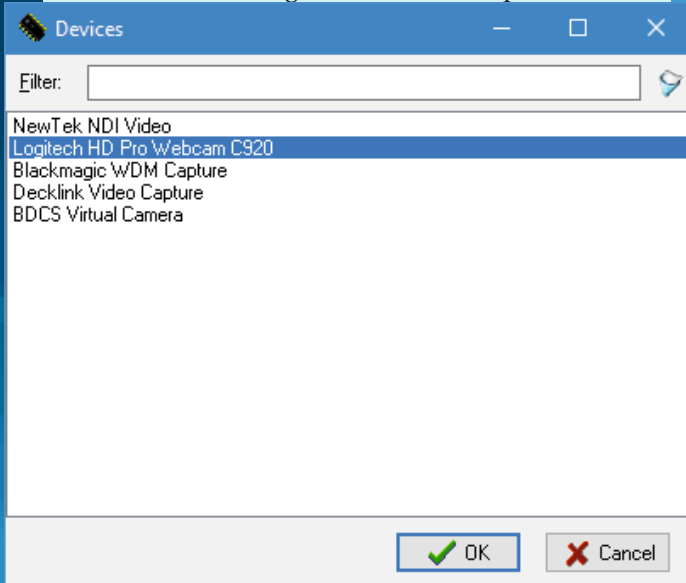
Connect the "Video" Output Pin of the **VLDSCapture1** to the "Video" Input Pin of the **VLImageDisplay1**:



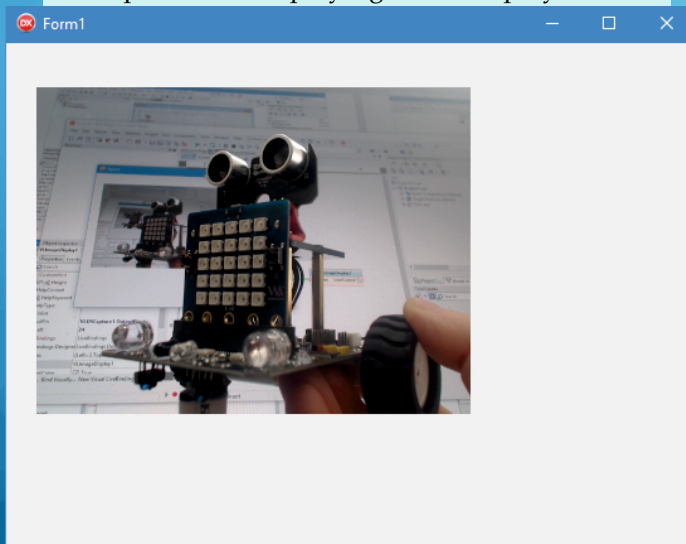
Click on the  button of the **VLDSCapture1** component:



In the Devices dialog, select a video capture device:



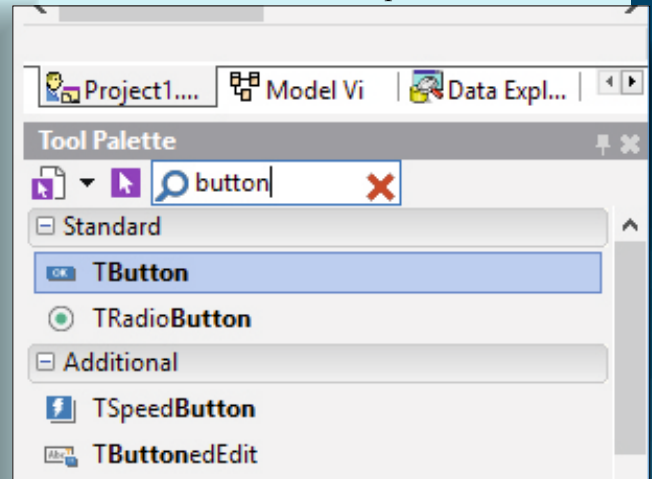
This component can capture from variety of devices, including video cameras, TV Tuners, and many others. Click on the OK button to close the dialog. Compile and run the application. You should see the captured video playing in the display:



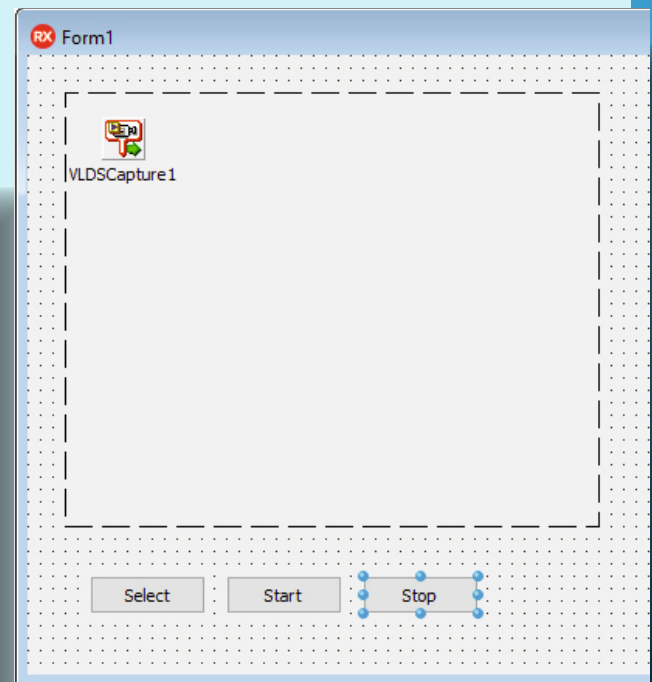
Close the application.

Now that you have learned how to capture the video, lets see how we can add some user interface to select the video source, and how to start and stop the component. Switch to the **Form Designer**.

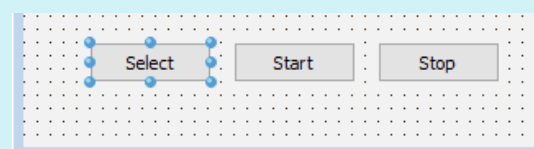
Type "button" in the **Tool Palette** search box, then select **TButton** from the palette:



Drop 3 of them on the form and in the Object Inspector set their Captions to "Select", "Start", and **"Stop"**:



Double click on the "Select" button to generate event handler:

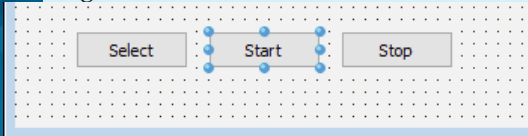


In the event handler write the following code:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    VLDSCapture1.VideoCaptureDevice.ShowDeviceSelctDialog();
end;
```

Switch to the **Form Designer**.

Double click on the "Start" button to generate event handler:



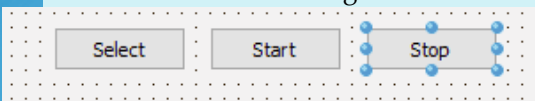
In the event handler write the following code:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    VLDSCapture1.Start();
end;
```

Alternatively you can use the Enabled property to start the capture:

```
VLDSCapture1.Enabled := True;
```

Switch to the Form Designer.



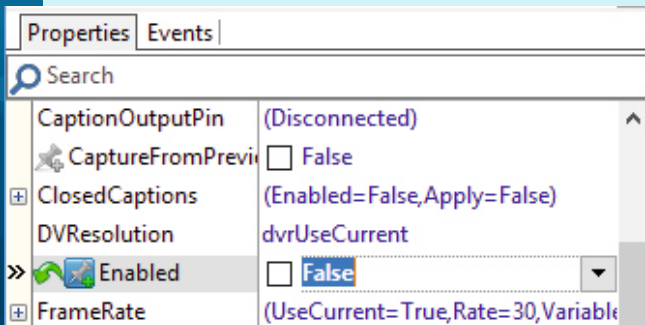
Double click on the "Stop" button to generate event handler: In the event handler write the following code:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    VLDSCapture1.Stop();
end;
```

Alternatively you can use the Enabled property to stop the capture:

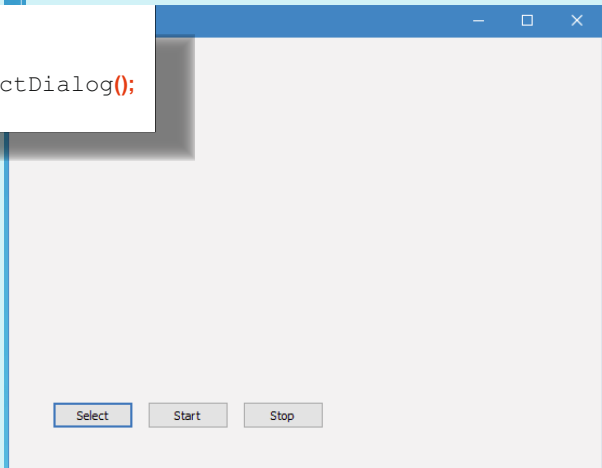
```
VLDSCapture1.Enabled := False;
```

Switch to the **Form Designer**, and select the **VLDSCapture1** component. In the **Object Inspector** set the "Enabled" property to "False":

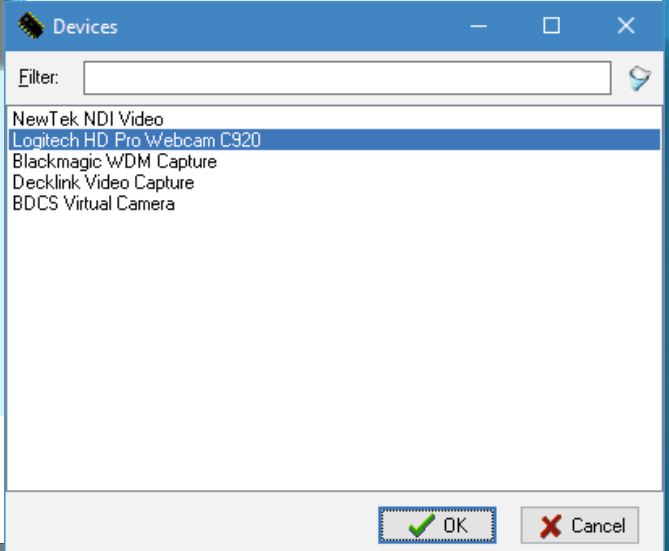


Compile and run the application.

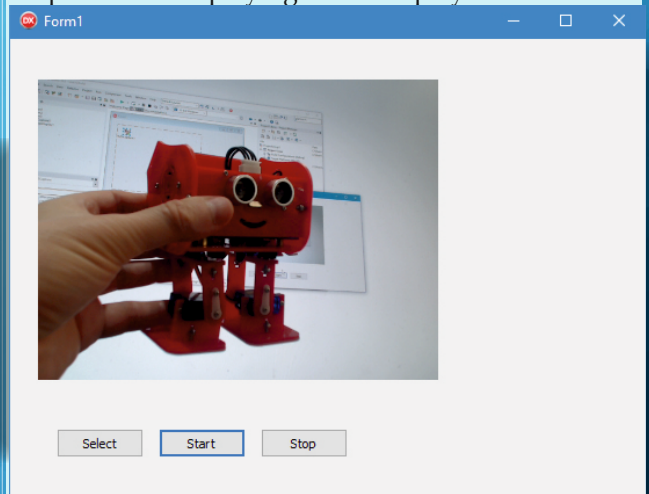
Click on the "Select" button



You should see the Devices selection dialog:



Select device and click OK to close the dialog. Click on the "Start" button. You should see the captured video playing in the display:

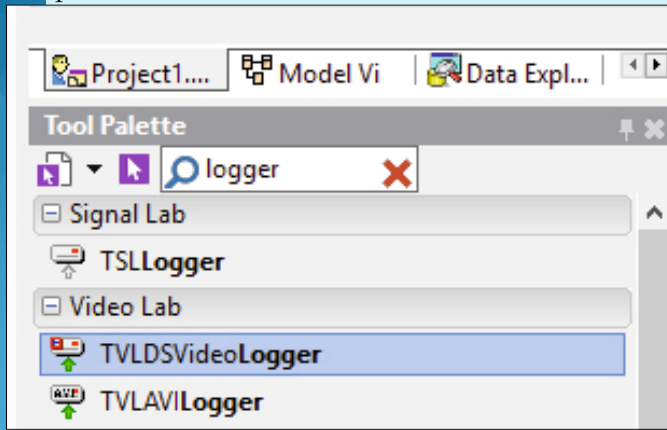


You can click on the "Stop" button to stop the capture.
Close the application.

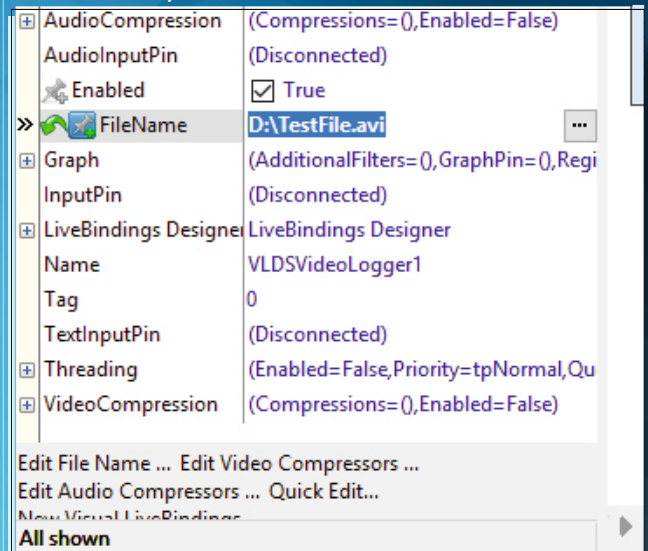
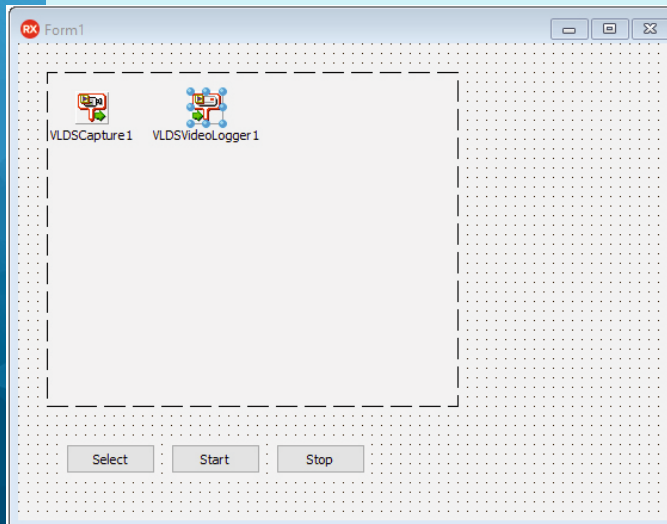
When capturing video usually we need to record it into file.
To do this we will add **Video Logger** component and connect the **Capture** component to it.

Switch to the **Form Designer**.

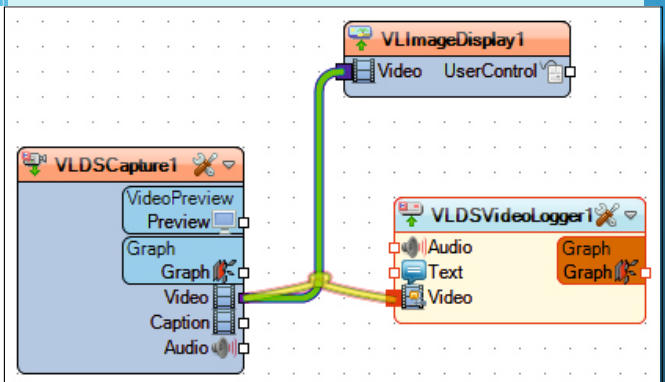
Type "logger" in the Tool Palette search box, then select **TVLDSVideoLogger** component from the palette:



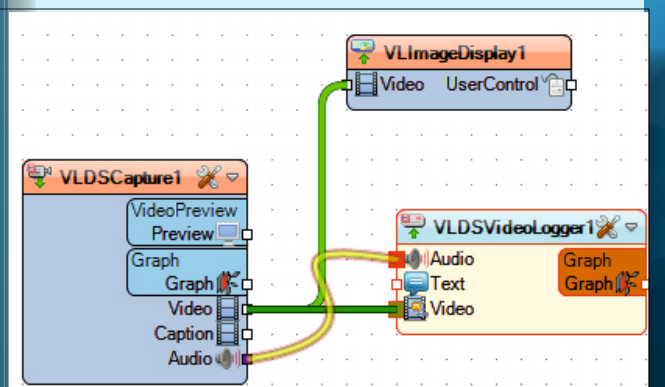
And drop it on the form.
In the Object Inspector, set the "FileName" property with the name of the file to record the video (In my case "D:\TestFile.avi"):



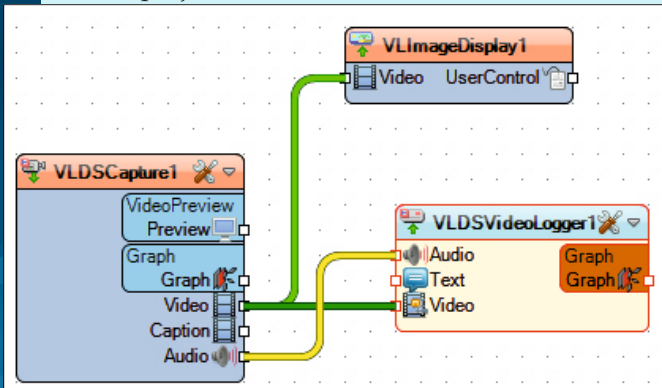
Switch to the "OpenWire" tab, and connect the "Video" Output Pin of the **VLDSVideoLogger1** to the "Video" Input Pin of the **VLDSVideoLogger1**:



Connect the "Audio" Output Pin of the **VLDSVideoLogger1** to the "Audio" Input Pin of the **VLDSVideoLogger1**:



Here is the complete **OpenWire Diagram** for this project:

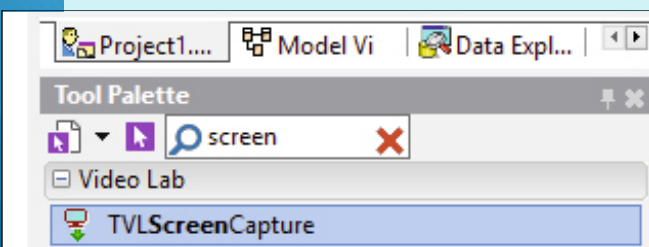


If you compile and run the application. You should see the captured video playing in the display, and the video will be recorded into the specified file.

The same application can be made using the older **TVLCapture** component, or the new cross-platform **TVLFMXVideoCapture** component. There are also many different options for **Video Logging** components using variety of video APIs and libraries.

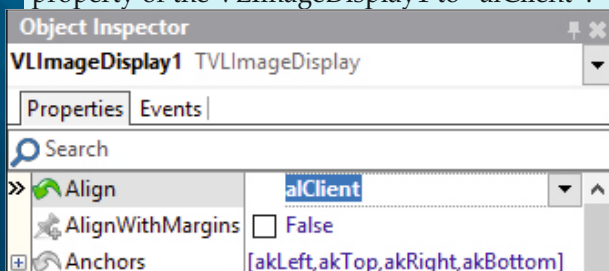
SCREEN CAPTURE

Now that you know how you can capture and record video from any DirectShow supported device such as Camera, or TV Tuner, I will show you how you can do Screen Capture into Video. Start a new VCL Form application. Type "screen" in the Tool Palette search box, then select TVLScreenCapture component from the palette:

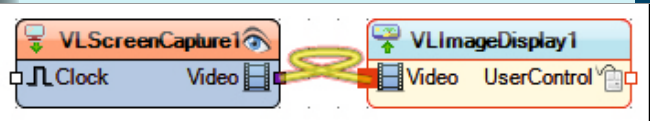


And drop it on the form.

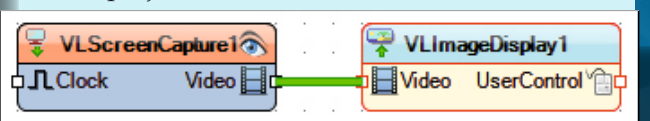
Next, add TVLImageDisplay, and set the "Align" property of the VLIImageDisplay1 to "alClient":



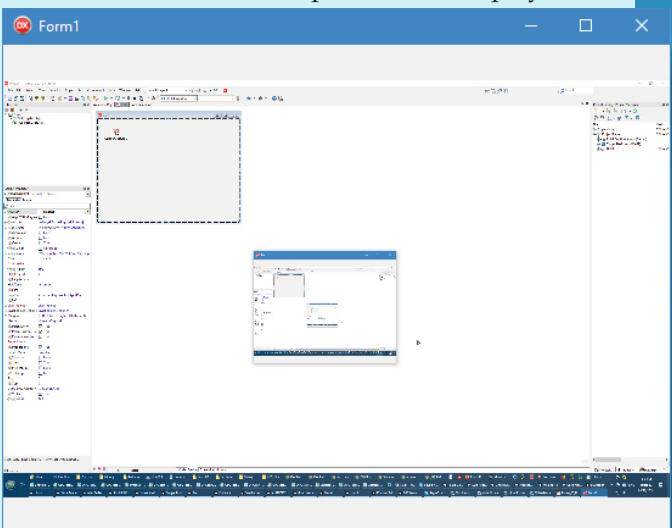
Switch to the "OpenWire" tab, and connect the "Video" Output Pin of the **VLScreenCapture1** to the "Video" Input Pin of the **VLIImageDisplay1**:



Here is the complete **OpenWire Diagram** for this project:



Compile and run the application. You should see the video of the screen capture in the display:

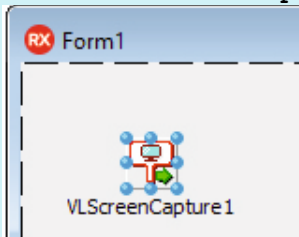


Close the application.

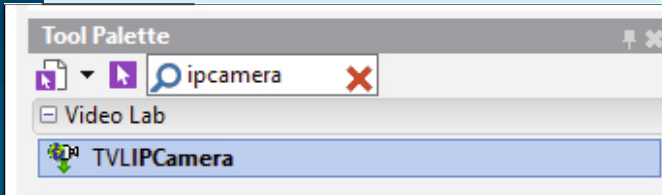
IP CAMERA AND WEBSTREAM CAPTURE

Now, that you already know how to capture from screen or directly connected to the computer devices, its time to show you how to capture from IP Camera connected on the network or RTSP web stream. Switch to the Form Designer.

Select the **VLScreenCapture1** component:



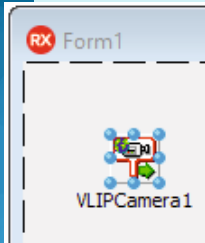
Delete the component. Type "ipcamera" in the Tool Palette search box, then select **TVLIPCamera** component from the palette:



And drop it on the form.

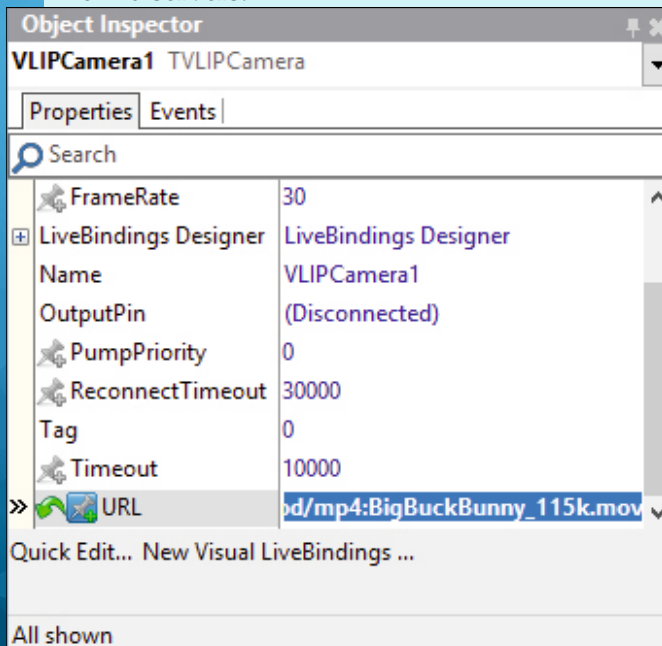
The component can capture video from most IP Cameras and **RTSP*** web streams.

*Real Time Streaming Protocol



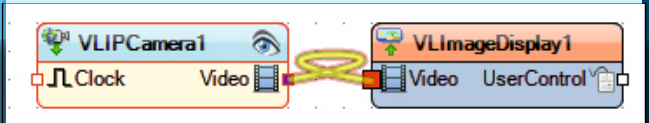
In the **Object Inspector** set the "URL" property to: `"rtsp://wowzaec2demo.streamlock.net/vod/mp4:BigBuckBunny_115k.mov"`

This is a free demo stream from the one of the Wowza servers:

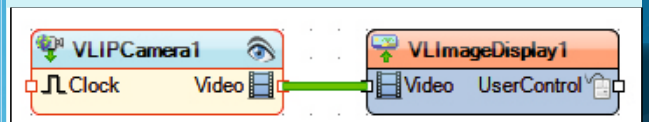


If you have an IP Camera, you can put the RTSP stream for it as described in the camera manual, or copy it from the camera management web interface. It will look something like this:
`rtsp://192.168.0.144/axis-media/media.amp?videocodec=h264`

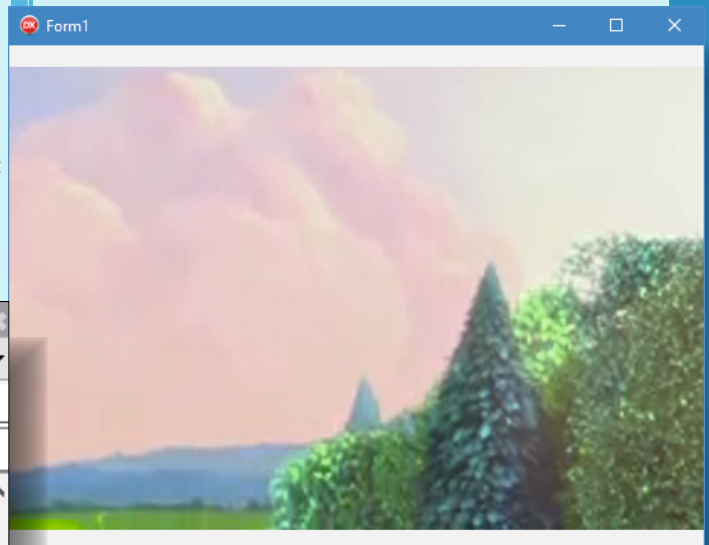
Switch to the "OpenWire" tab. Connect the "Video" Output Pin of the **VLIPCamera1** to the "Video" Input Pin of the **VLImageDisplay1**:



Here is the complete **OpenWire Diagram** for this project:



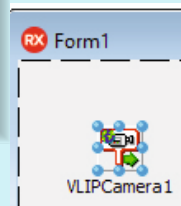
Compile and run the application. You should see the video stream playing in the display:



Close the application.

Now, that you already know how to capture from IP Camera or **RTSP** web stream, I will show you how you can receive video from **ASF/WMV*** video stream or web hosted file. Switch to the Form Designer.

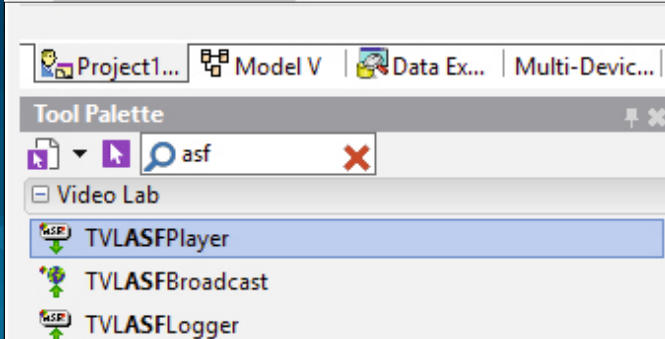
Select the **VLIPCamera1** component:



*Advanced Systems Format / Windows Media Video

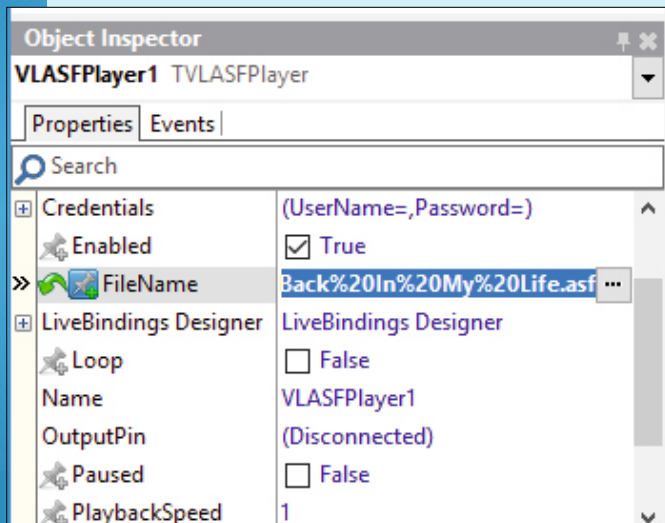
Delete the component.

Type "asf" in the **Tool Palette** search box, then select **TVLASFPlayer** component from the palette:

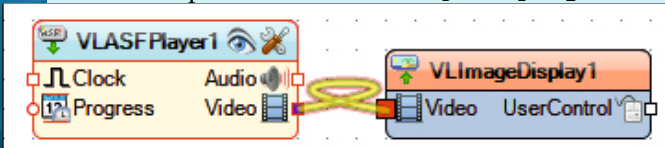


And drop it on the form.

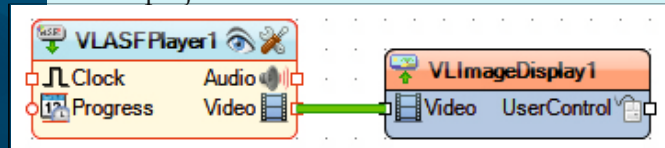
In the **Object Inspector** set the value of the "FileName" property to the URL of the video stream that you want to play, as example:
<https://samples.ffmpeg.org/asf-wmv/Alice%20Deejay%20-%20Back%20In%20My%20Life.asf>



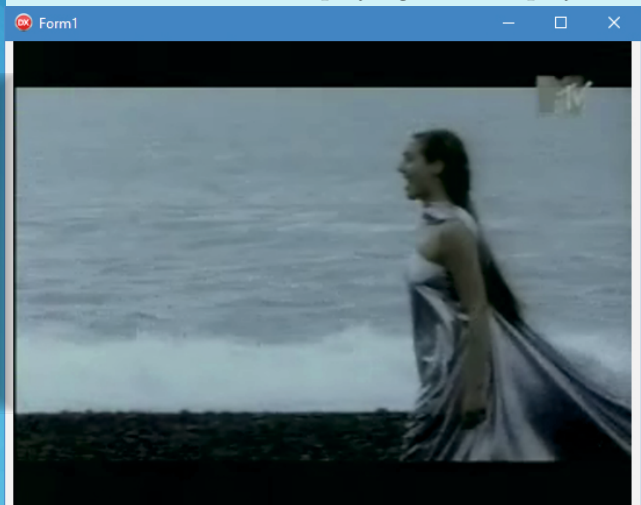
Switch to the "OpenWire" tab. Connect the "Video" Output Pin of the **VLASFPPlayer1** to the "Video" Input Pin of the **VLImageDisplay1**:



Here is the complete **OpenWire Diagram** for this project:



Compile and run the application.
 You should see the video playing in the display:



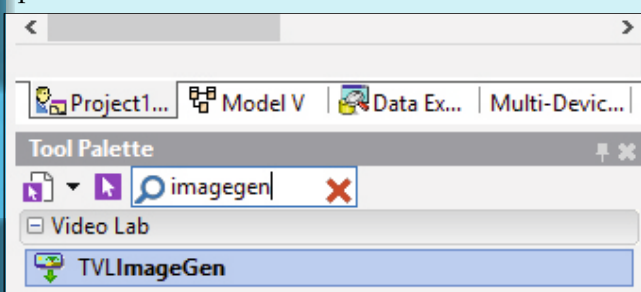
Close the application.

GENERATE A VIDEO

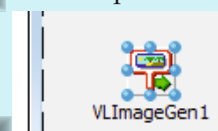
Now, that you already know how to capture from variety of devices and online streams, it is time to show you how you can generate your own video from bitmaps or by drawing on canvas. First we will use image generator with fixed image as video source. Switch to the Form Designer.

Select the **VLASFPPlayer1** and delete the component.

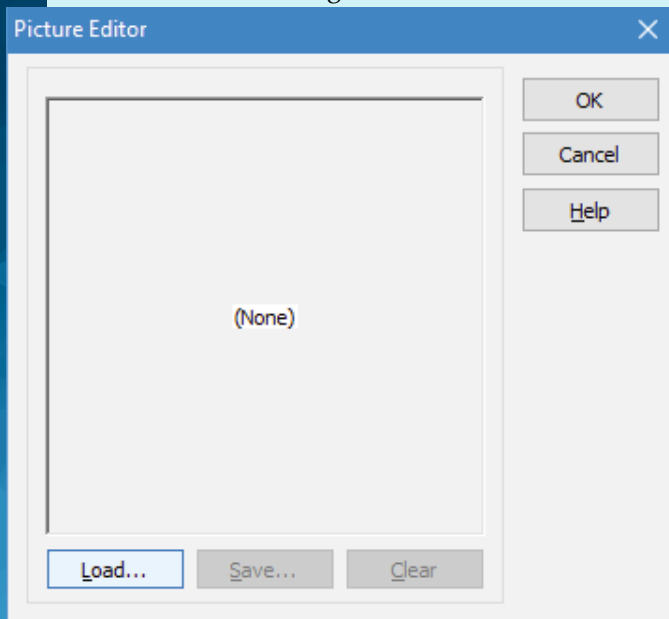
Type "imagegen" in the **Tool Palette** search box, then select **TVLImageGen** component from the palette:



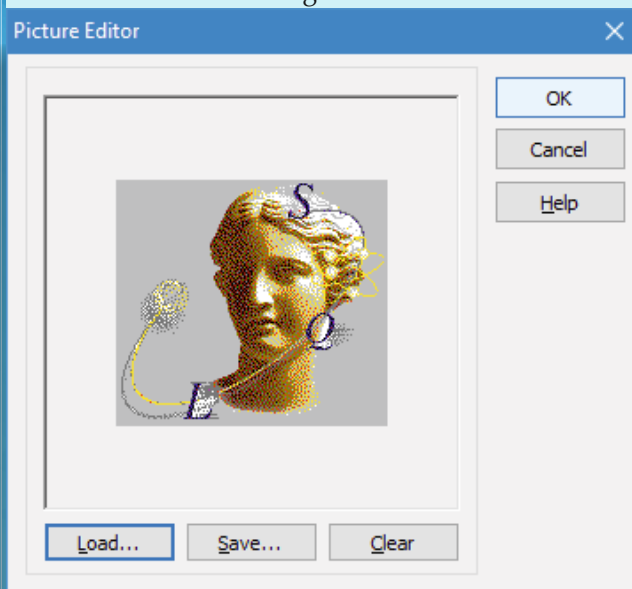
And drop it on the form:



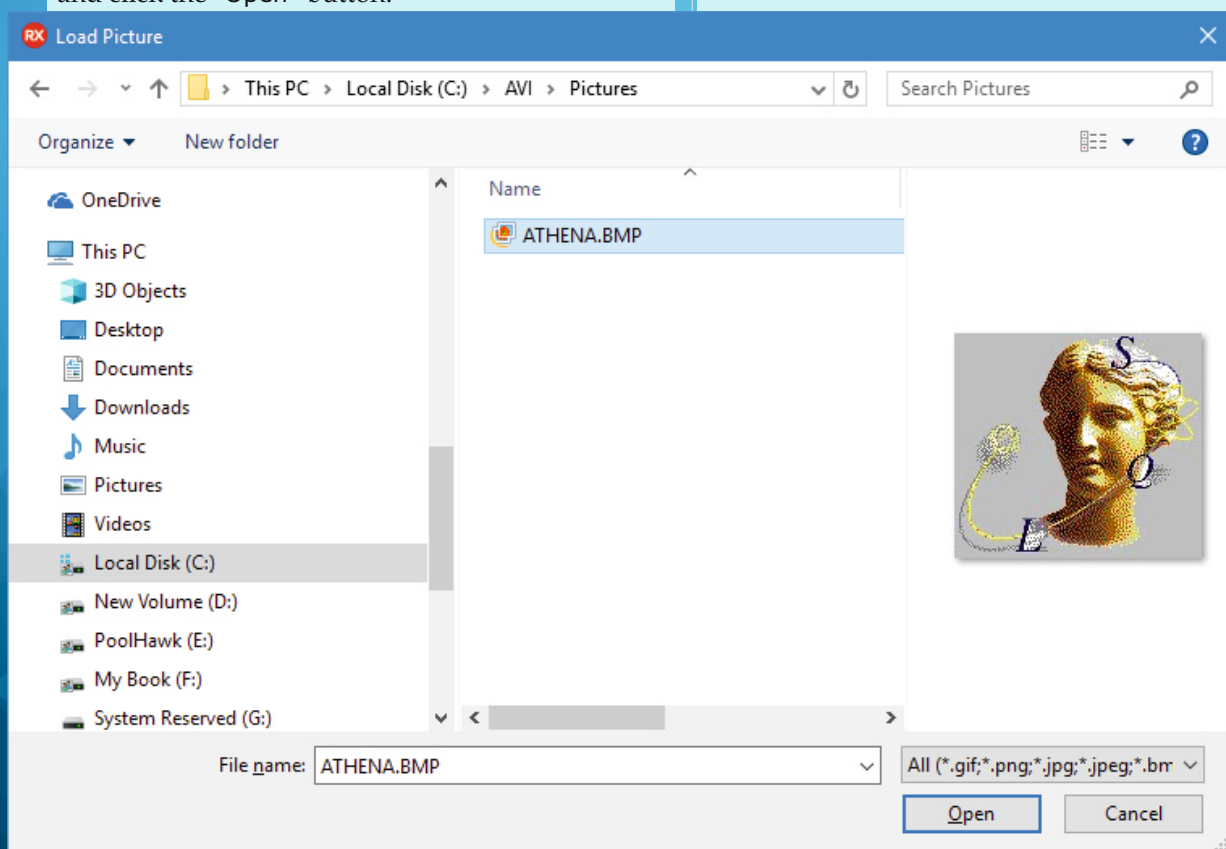
Double click on the component to open the "Picture Editor" dialog. Click on the "Load...":



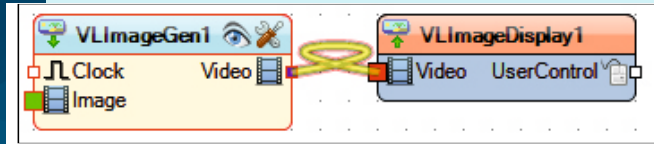
Click on the "OK" button of the "Picture Editor" Dialog:



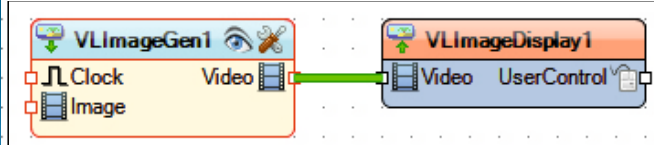
In the "Load Picture" dialog select bitmap file, and click the "Open" button:



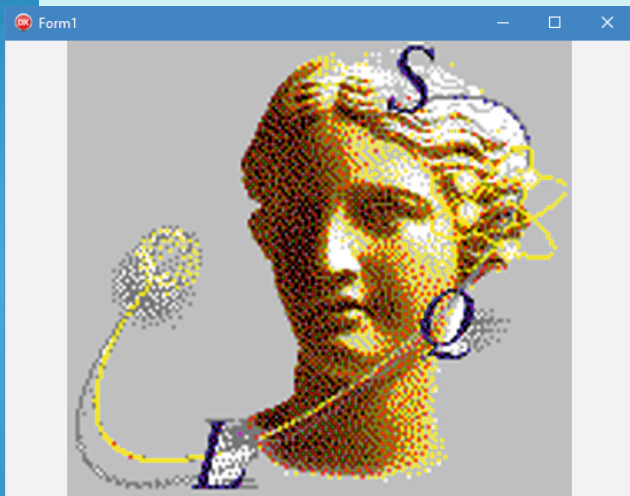
Switch to the "OpenWire" tab. Connect the "Video" Output Pin of the **VLImageGen1** to the "Video" Input Pin of the **VLImageDisplay1**:



Here is the complete **OpenWire Diagram** for this project:



Compile and run the application. You should see the image in the display:



Close the application.

Now it's time to try generating video from Delphi code.

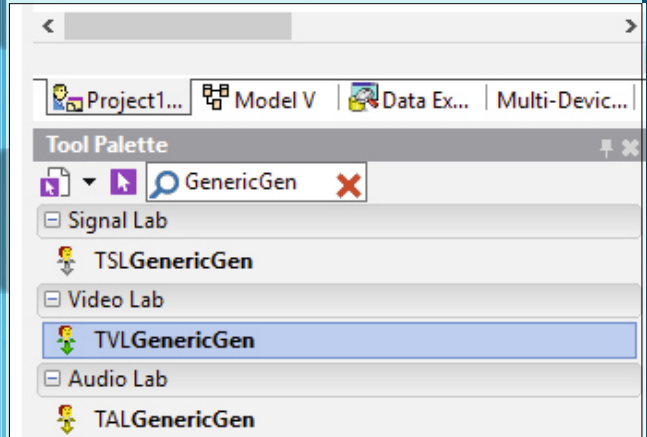
There are 2 components that are suitable for this - **TVLImageGen** and **TVLGenericFilter**. I already showed you in the previous articles how to use the **TVLGenericFilter** to process video. It is also capable of working as a video source. The difference between the **TVLImageGen** and **TVLGenericFilter** is that the **TVLImageGen** will automatically generate periodic **OnGenerate** event, where you can place your Delphi code. When using **TVLGenericFilter** you will need to write your own code to send data through the component usually in a loop, **TTimer** or other event.

We will start with the **TVLImageGen**.

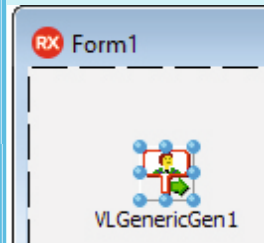
Switch to the **Form Designer**.

Select the **VLImageGen1** and delete the component.

Type "genericgen" in the **Tool Palette** search box, then select **TVLGenericGen** component from the palette:



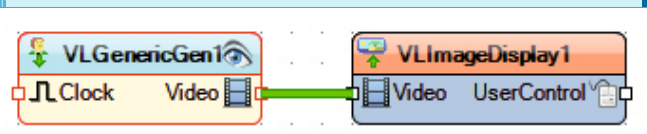
And drop it on the form:



Switch to the "OpenWire" tab. Connect the "Video" Output Pin of the **VLGenericGen1** to the "Video" Input Pin of the **VLImageDisplay1**:



Double click on the **VLGenericGen1** to generate the **OnGenerate** event handler:



Add the following unit to the uses clause:

```
uses
  VCL.Mitov.ImageBuffer;
```

and in the **VLGenericGen1Generate** event handler add the following code:

```
procedure TForm1.VLGenericGen1Generate(Sender: TObject;
  var OutBuffer: IVLImageBuffer; var Populated, Finished: Boolean);
var
  ABitmap: TBitmap;
begin
  ABitmap := TBitmap.Create();
  ABitmap.SetSize(240, 180);
  ABitmap.Canvas.Brush.Color := clWhite;
  ABitmap.Canvas.FillRect( TRect.Create(0, 0, 240, 180 ));

  ABitmap.Canvas.Brush.Color := clRed;
  ABitmap.Canvas.Pen.Color := clBlue;
  ABitmap.Canvas.Ellipse( 0, 0, 100, 100 );

  OutBuffer.Access.FromBitmap( ABitmap );

  ABitmap.DisposeOf();
end;
```

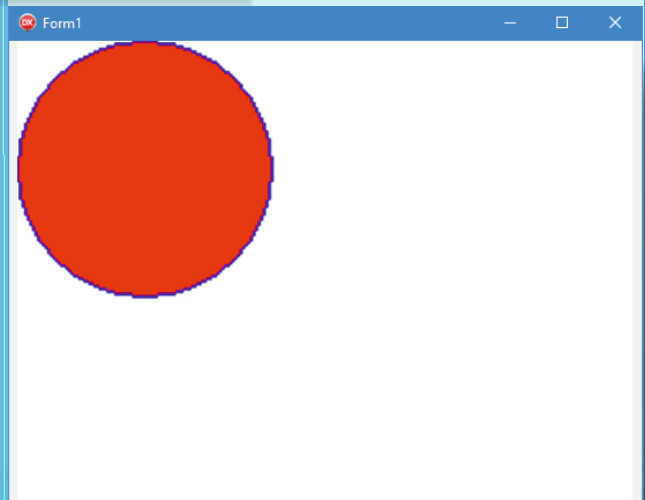
Here we create a bitmap, set its size to 240, 180, draw on the bitmap, and finally call:

```
OutBuffer.Access.FromBitmap( ABitmap );
```

to assign the bitmap to the video buffer.

If you need to generate frames with different size, you should also set the "Width", and "Height" of the "ImageSize" property of the **VLGenericGen1**.

Compile and run the application. You should see the circle in the display:



Close the application.

Here is the complete source for this project:

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Mitov.VCLTypes, VCL.LPCControl, SLControlCollection,
  VLCommonDisplay, VLImageDisplay, Mitov.Types, SLCommonGen, VLCommonGen, VLBasicGenericGen,
  VLGenericGen;

type
  TForm1 = class(TForm)
    VLGenericGen1: TVLGenericGen;
    VLImageDisplay1: TVLImageDisplay;
    procedure VLGenericGen1Generate(Sender: TObject;
      var OutBuffer: IVLImageBuffer; var Populated, Finished: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses
  VCL.Mitov.ImageBuffer;

procedure TForm1.VLGenericGen1Generate(Sender: TObject;
  var OutBuffer: IVLImageBuffer; var Populated, Finished: Boolean);
var
  ABitmap: TBitmap;
begin
  ABitmap := TBitmap.Create();
  ABitmap.SetSize(240, 180);
  ABitmap.Canvas.Brush.Color := clWhite;
  ABitmap.Canvas.FillRect(TRect.Create(0, 0, 240, 180));

  ABitmap.Canvas.Brush.Color := clRed;
  ABitmap.Canvas.Pen.Color := clBlue;
  ABitmap.Canvas.Ellipse(0, 0, 100, 100);

  OutBuffer.Access.FromBitmap(ABitmap);

  ABitmap.DisposeOf();
end;

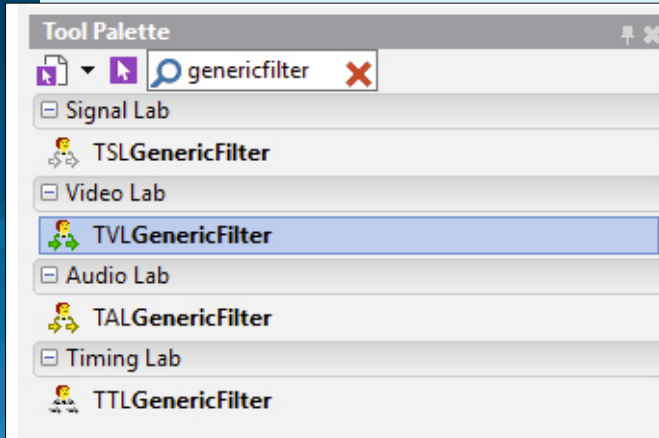
end.
```

Next we will do the same project by using
TVLGenericFilter.

Select the **VLGenericGen1** and
delete the component.

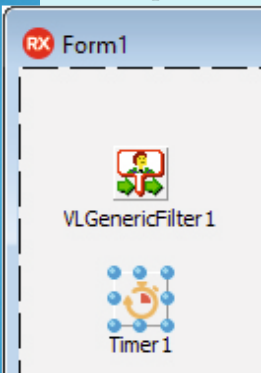


Type **"genericfilter"** in the **Tool Palette** search box, then select **TVLGenericFilter** component from the palette:



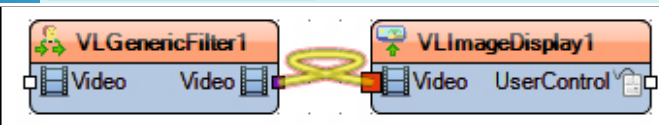
And drop it on the form.

Type **"timer"** in the **Tool Palette** search box, then select **TTimer** component from the palette:
And drop it on the form:

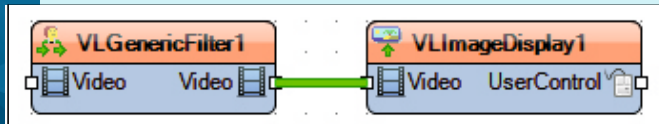


Switch to the **"OpenWire"** tab.

Connect the **"Video"** Output Pin of the **VLGenericFilter1** to the **"Video"** Input Pin of the **VLImageDisplay1**:



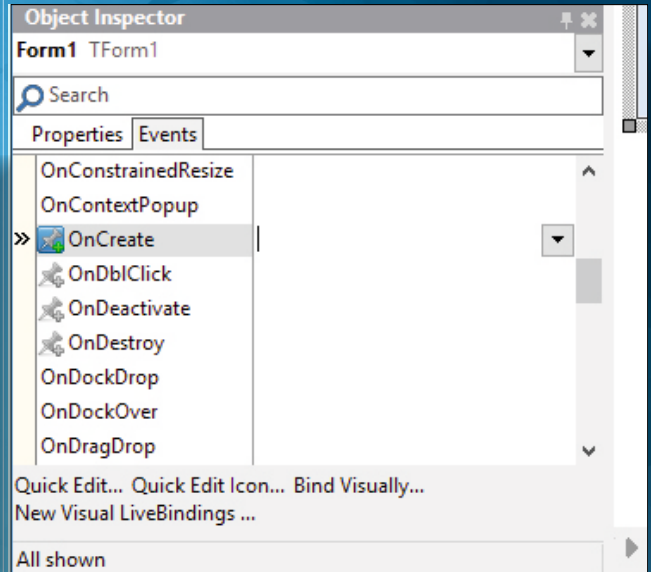
Here is the complete **OpenWire Diagram** for this project:



Switch to the **Form Designer**.

Select the Form.

Switch to the **Events** tab of the **Object Inspector**. Double click on the **OnCreate** event to generate event handler:



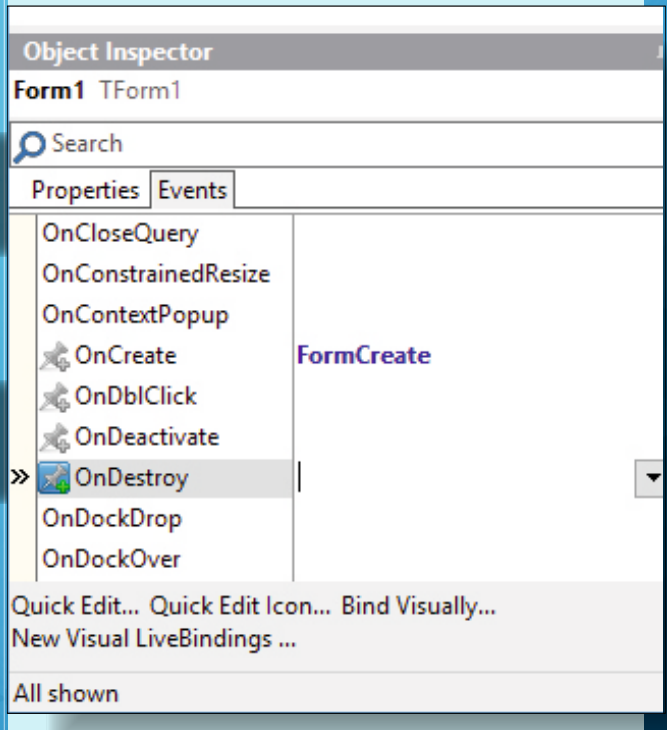
In the event handler add the following code:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    VLGenericFilter1.SendStartCommand(240,180,1000);
end;
```

this command will start the video streaming with frame size 240 by 180 pixels, and 1000 milliseconds between frames.

Switch to the **Form Designer**.

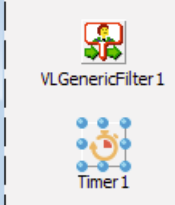
Double click on the **OnDestroy** event to generate event handler:



In the event handler add the following code:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
    VLGenericFilter1.SendStopCommand();
end;
```

This command will stop the video streaming. Switch to the **Form Designer**. Double-click on the **Timer1** component to generate the event handler:



In the event handler add the following code:

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
    ABitmap: TBitmap;
begin
    ABitmap := TBitmap.Create();
    ABitmap.SetSize( 240, 180 );
    ABitmap.Canvas.Brush.Color := clWhite;
    ABitmap.Canvas.FillRect( TRect.Create( 0, 0, 240, 180 ))

    ABitmap.Canvas.Brush.Color := clRed;
    ABitmap.Canvas.Pen.Color := clBlue;
    ABitmap.Canvas.Ellipse( 0, 0, 100, 100 );

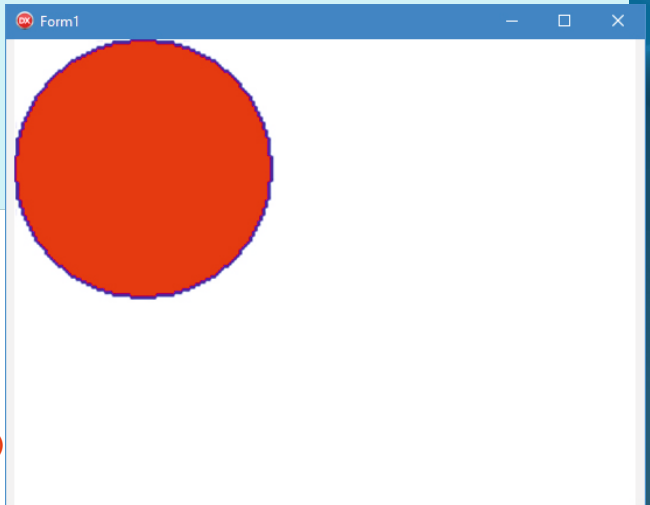
    VLGenericFilter1.SendData(
        TVLImageBuffer.CreateBmp( ABitmap ));

    ABitmap.DisposeOf();
end;
```

The code is almost identical to the code in the previous sample, except that here instead of assigning the bitmap to buffer, we will create a buffer from the bitmap, and send it to the **VLGenericFilter1**:

```
VLGenericFilter1.SendData(
    TVLImageBuffer.CreateBmp( ABitmap ));
```

Compile and run the application. After about 1 second, when the timer executes its event, you should see the circle in the display:



Close the application.

Here is the complete source code for this project:

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Mitov.VCLTypes, VCL.LPControl, SLControlCollection,
  VLCommonDisplay, VLIImageDisplay, Mitov.Types, VLDSCapture, Vcl.StdCtrls, LComponent,
  SLCommonFilter, VLCommonLogger, VLDSVideoLogger, SLCommonGen, VLCommonGen, VLScreenCapture,
  VLIPCamera, MLWMFBaseComponent, MLASFPlayer, VLASFPlayer, VLIImageGen, VLBasicGenericGen,
  VLGenericGen, Vcl.ExtCtrls, VLCommonFilter, VLBasicGenericFilter, VLGenericFilter;

type
  TForm1 = class(TForm)
    VLIImageDisplay1: TVLIImageDisplay;
    VLGenericFilter1: TVLGenericFilter;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses
  VCL.Mitov.ImageBuffer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  VLGenericFilter1.SendStartCommand( 240, 180, 1000 );
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  VLGenericFilter1.SendStopCommand();
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var  ABitmap: TBitmap;

begin
  ABitmap := TBitmap.Create();
  ABitmap.SetSize( 240, 180 );
  ABitmap.Canvas.Brush.Color := clWhite;
  ABitmap.Canvas.FillRect( TRect.Create( 0, 0, 240, 180 ));

  ABitmap.Canvas.Brush.Color := clRed;
  ABitmap.Canvas.Pen.Color := clBlue;
  ABitmap.Canvas.Ellipse( 0, 0, 100, 100 );

  VLGenericFilter1.SendData( TVLIImageBuffer.CreateBmp( ABitmap ));

  ABitmap.DisposeOf();
end;

end.
```



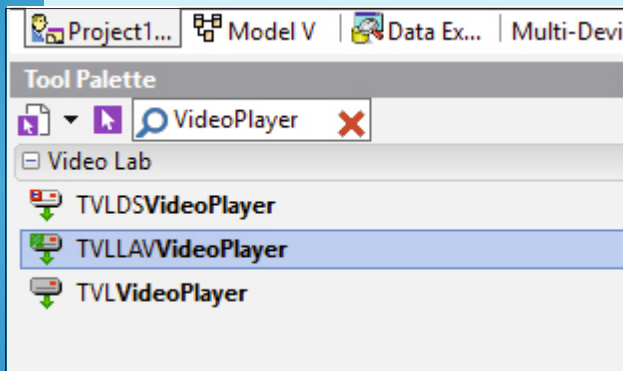
VIDEO BROADCASTING

I already showed you how to record the generated or captured video. Now I will show you how you can broadcast it over local area network or over ASF Internet stream. We will start with local arena broadcast using NDI - NewTek's Network Device Interface technology. NDI allows devices to send video and audio streams between them over local area network using UDP.

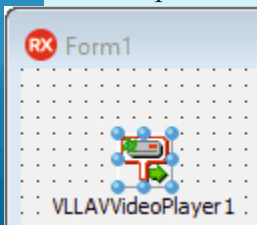
Start a new **VCL** Form application.

For this application I will use a different video player - **TVLLAVVideoPlayer**. It uses the **FFMpeg** library to decode the video, and can support large number of video formats. You can use any of the Video Players included in **VideoLab**.

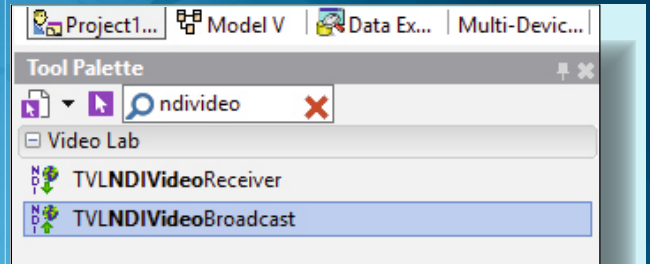
Type "VideoPlayer" in the Tool Palette search box, then select **TVLLAVVideoPlayer** component from the palette:



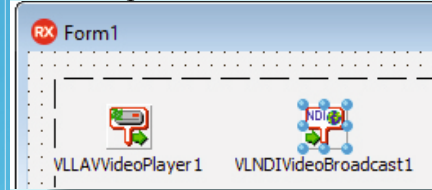
And drop it on the Form.



Type "imagedi" in the **Tool Palette** search box, then select **VLLAVVideoPlayer1** component from the palette, and drop it on the Form. Type "ndivideo" in the **Tool Palette** search box, then select **TVLNDDIVideoBroadcast** component from the palette:

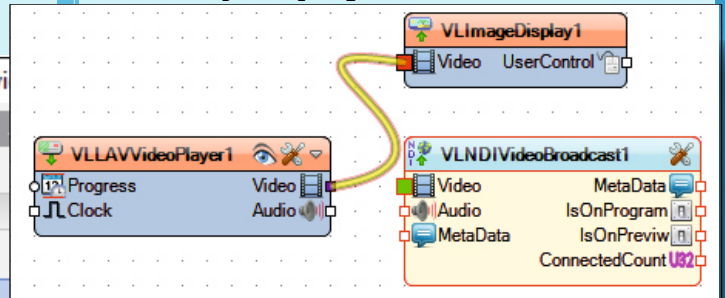


And drop it on the Form:

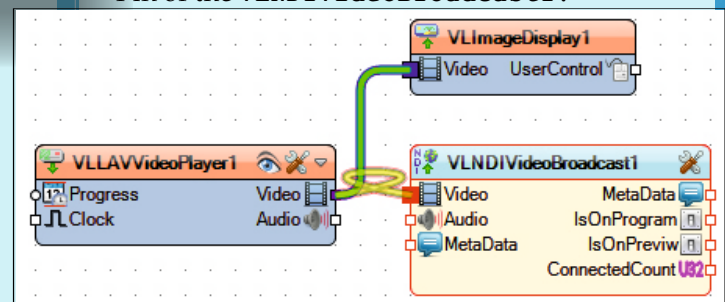


Switch to the "OpenWire" tab.

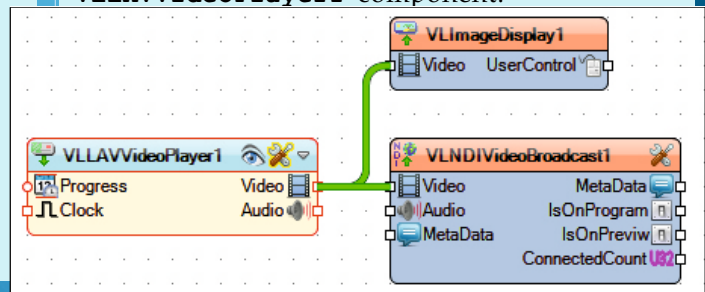
Connect the "Video" Output Pin of the **VLLAVVideoPlayer1** to the "Video" Input Pin of the **VLImageDisplay1**:



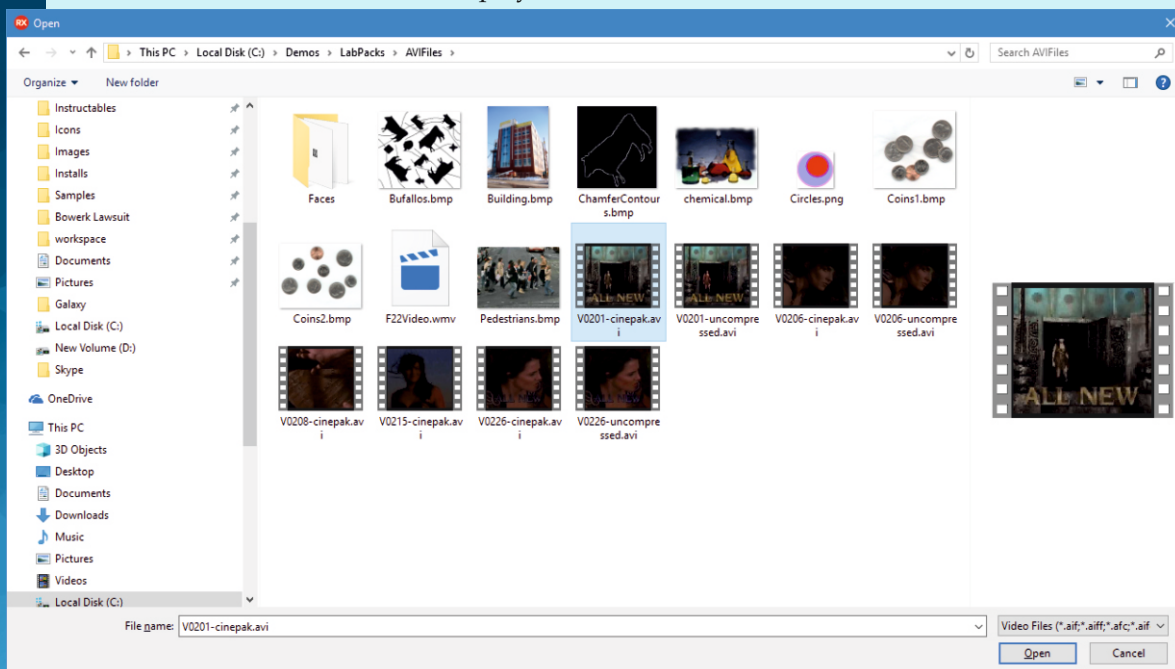
Connect the "Video" Output Pin of the **VLLAVVideoPlayer1** to the "Video" Input Pin of the **VLNDDIVideoBroadcast1**:



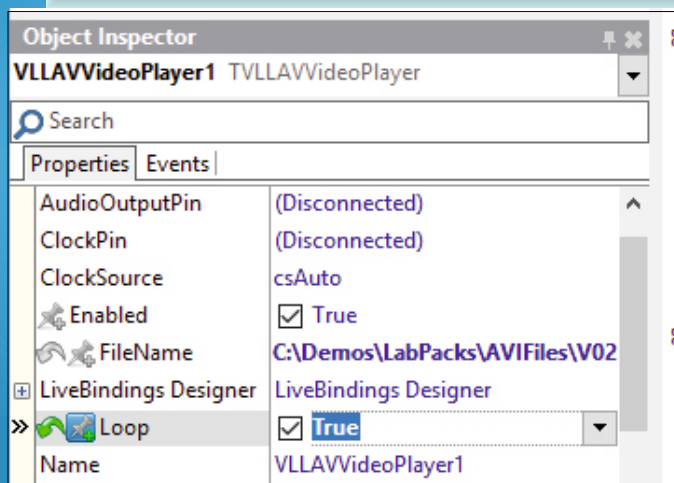
Click on the  button of the **VLLAVVideoPlayer1** component:



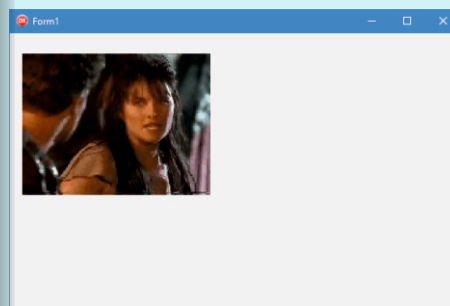
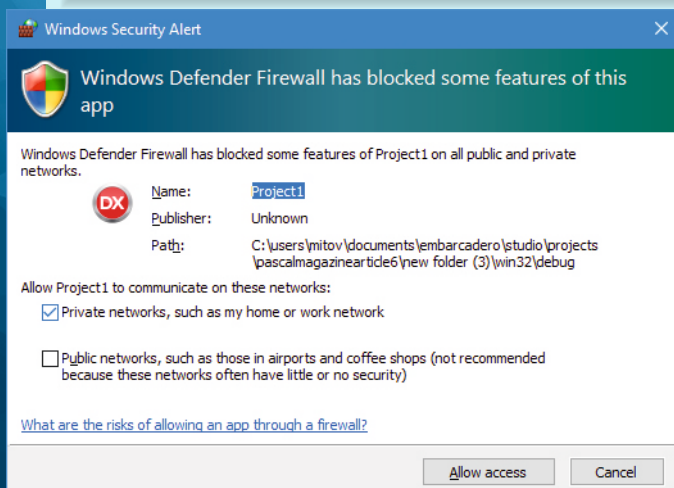
In the File **Open** Dialog select file to play, and click the "Open" button:



In the **Object Inspector** set the "Loop" property to True:

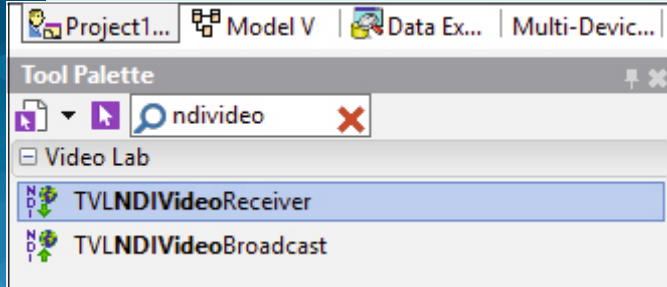


Compile and run the application. You may see a Firewall alert since the application will try to broadcast over the network. Click "Allow access":

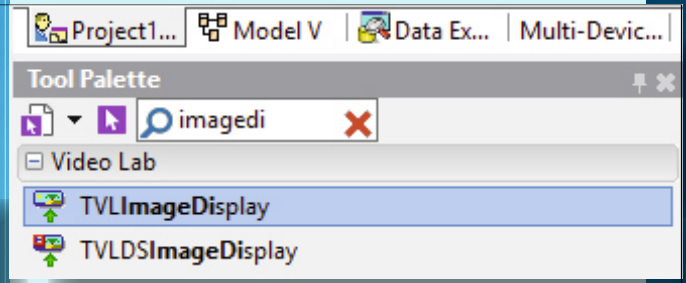


You should see the video playing in the display.

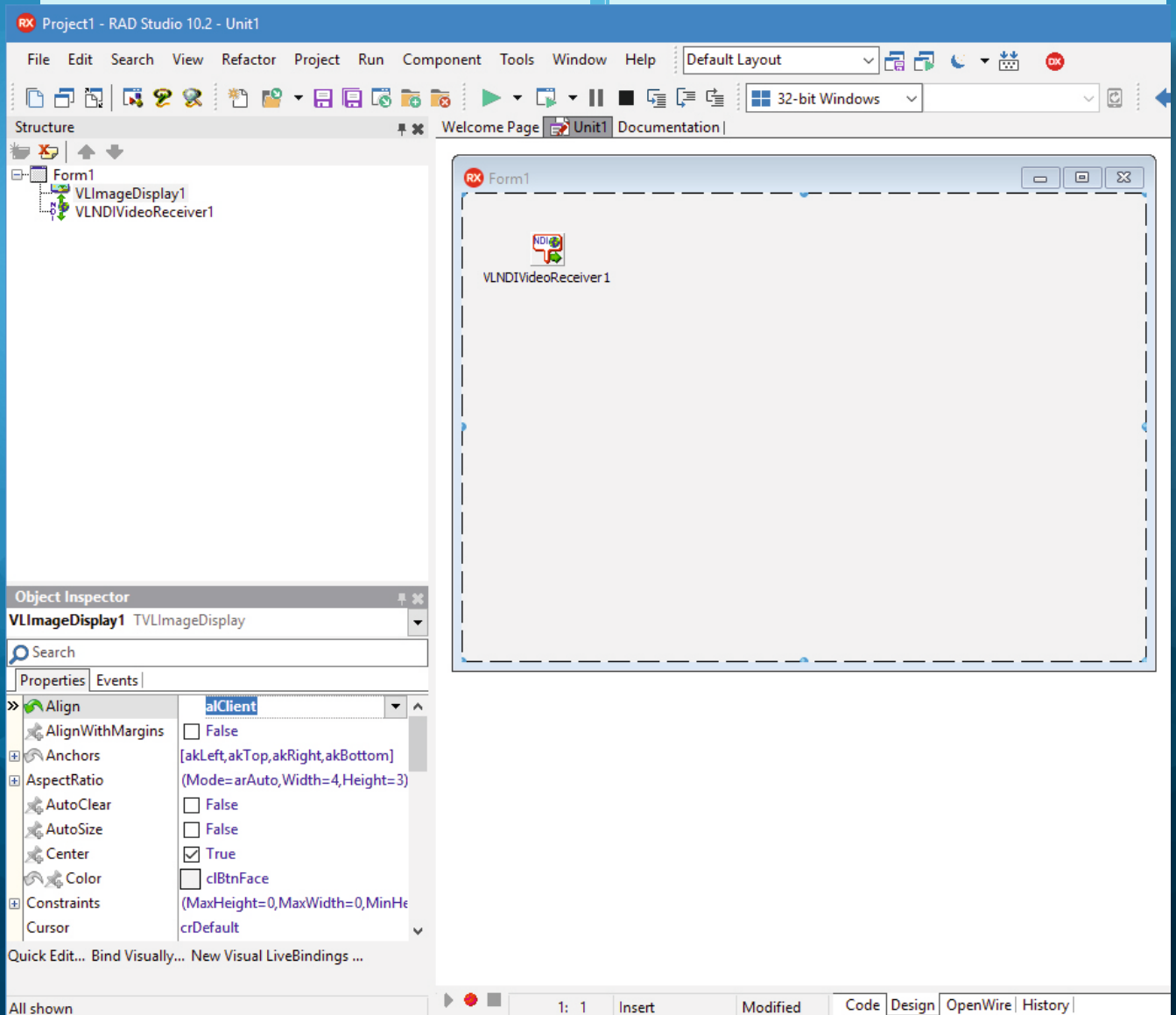
Leave the application running.
To receive the video that the application is sending, we will create a second application in Delphi.
Start a new **VCL** Form application.
Type "ndivideo" in the **Tool Palette** search box, then select **TVLNDIVideoReceiver** component from the palette:



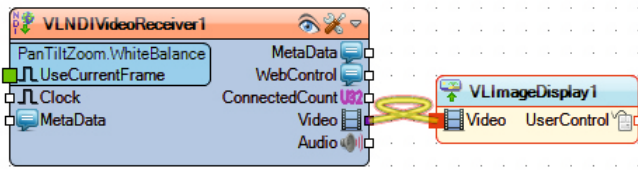
And drop it on the form.
Type "imagedi" in the Tool Palette search box, then select **TVLImageDisplay** from the palette:



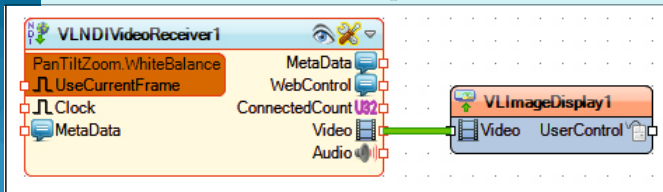
And drop it on the form.
In the Object Inspector set the "Align" property of the **VLImageDisplay1** to "alClient":



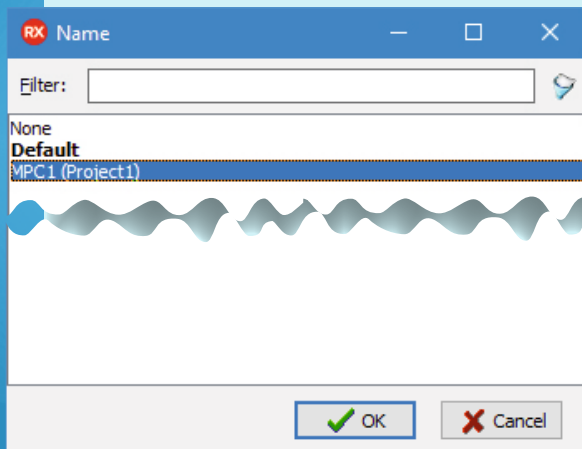
Connect the "Video" Output Pin of the **VLNDIVideoReceiver1** to the "Video" Input Pin of the **VLImageDisplay1**



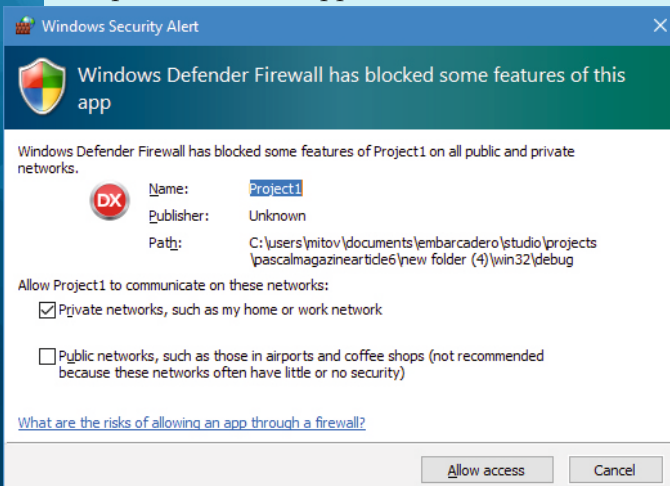
Click on the button of the **VLNDIVideoReceiver1** component:



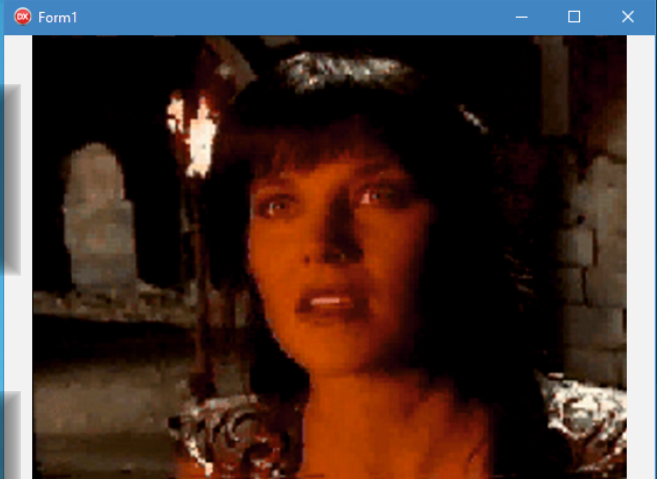
In the dialog select the video stream to play. In this case the default video stream of your own computer, and click the OK button:



Compile and run the application.



After few seconds you should see the video sent from the other application on the display:



Close both applications.

NDI* is good for broadcasting over local network, but not when you need to broadcast over internet.

For internet broadcast you can use the **TVLASFBroadcast** component instead.

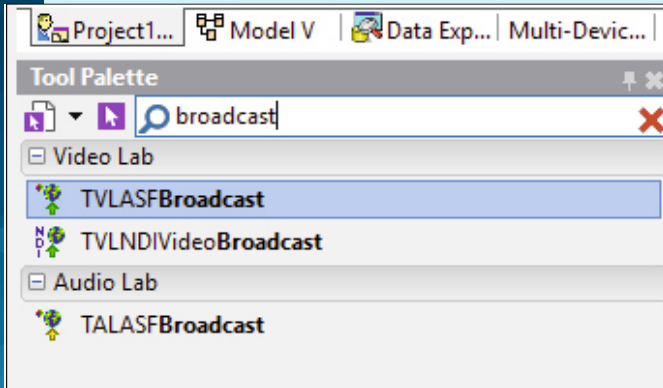
Open the **NDI Video Broadcast** project that we created earlier.

Select the **VLNDIVideoBroadcast1** and delete the component.

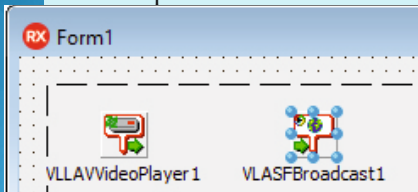
*NDI = Network Device Interface

You may see a Firewall alert since the application will try to broadcast over the network. Click "Allow access".

Type "broadcast" in the **Tool Palette** search box, then select **TVLASFBroadcast** component from the palette:



And drop it on the form:



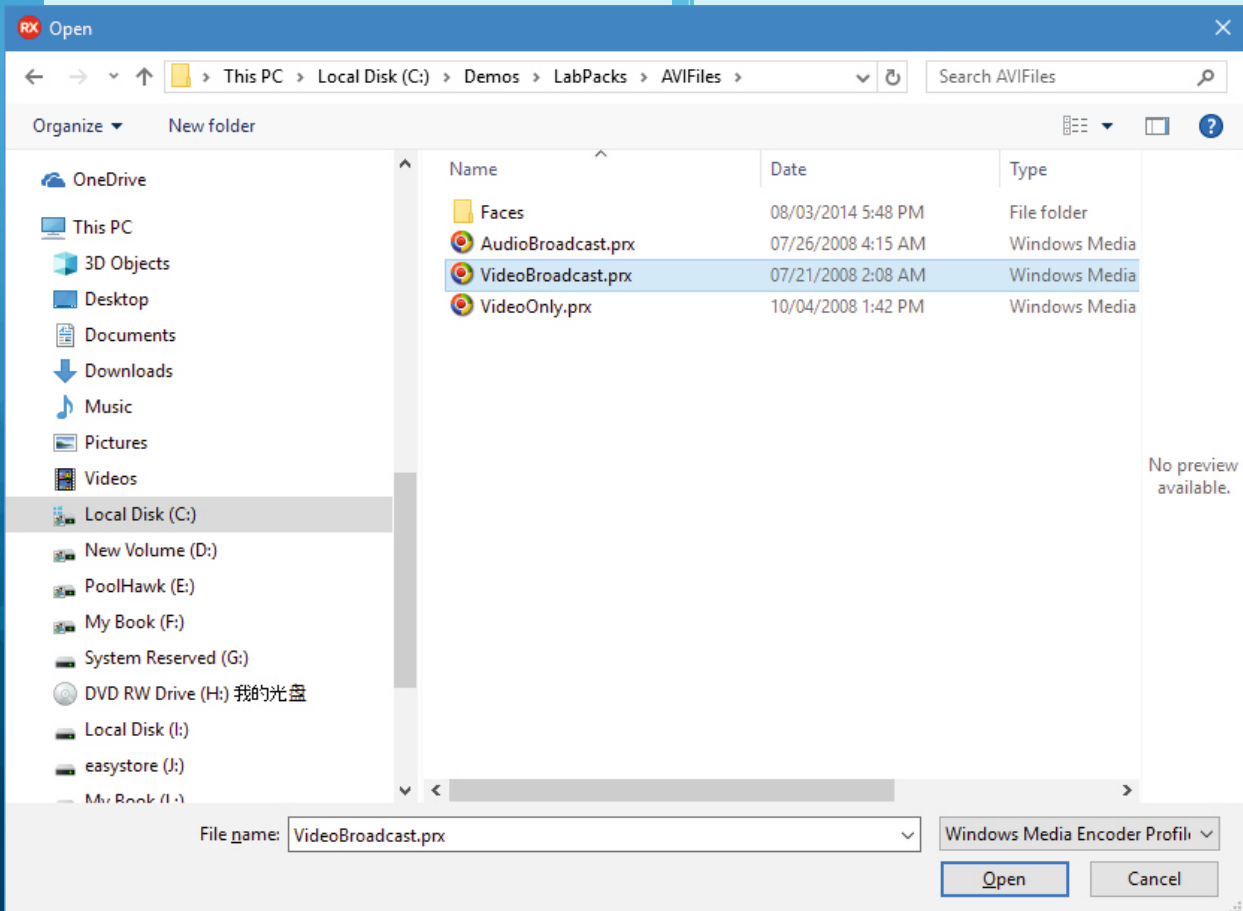
Double-click on the component to open the Profile Editor:

Select a profile file, and click on the "Open" button.

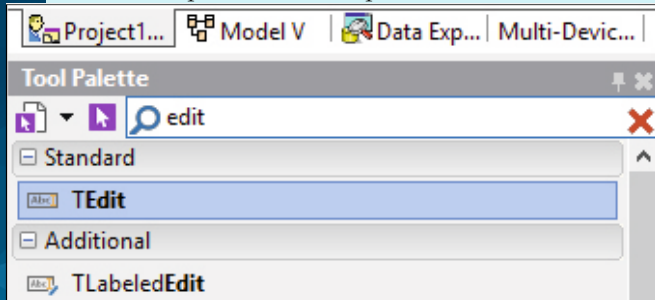
The **VideoLab** installation includes a profile file for compressing video with 240 by 180 pixels frame size, that you can use if the video you want to broadcast is of this size:

C:\Program Files (x86)\Embarcadero\Studio\19.0\LabPacks\Demos\AVIFiles\VideoBroadcast.prx
Otherwise you will need to create your own profile. **Microsoft** has created an **ASF** profile editor, and it can be found on the web at a number of download sites.

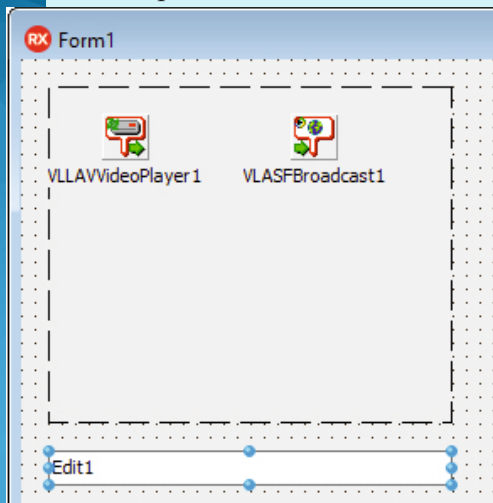
Now we will add an edit box to show the local broadcasting URL.



Type "edit" in the **Tool Palette** search box, then select **TEdit** component from the palette:



And drop it on the form:

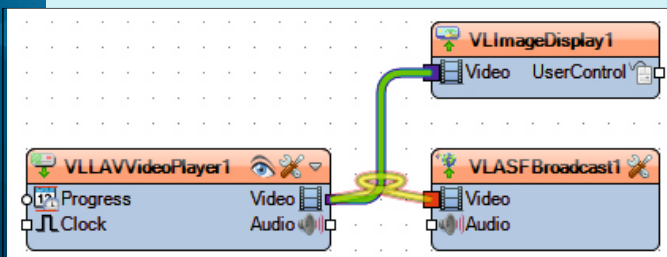


Double-click on the form to generate the **OnCreate** event handler:

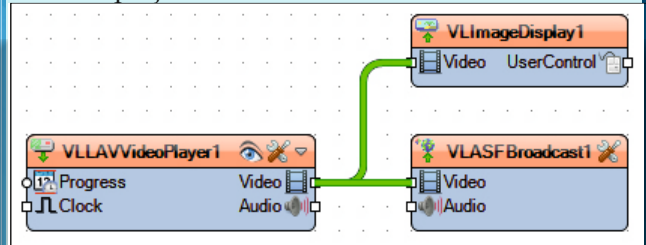
In the event handler add the following code:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Text := VLASFBCast1.LocalBroadcast.HostURL;
end;
```

Switch to the "OpenWire" tab.
Connect the "Video" Output Pin of the **VLLAVVideoPlayer1** to the "Video" Input Pin of the **VLASFBCast1**:

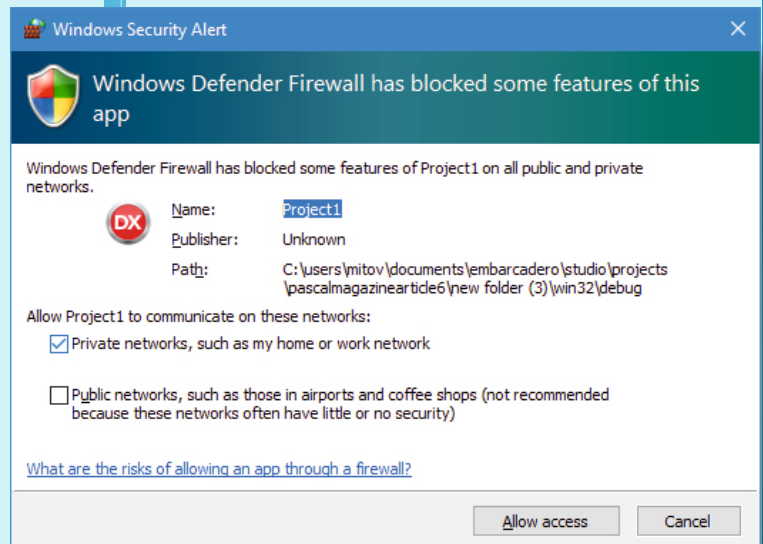


Here is the complete **OpenWire Diagram** for this project:

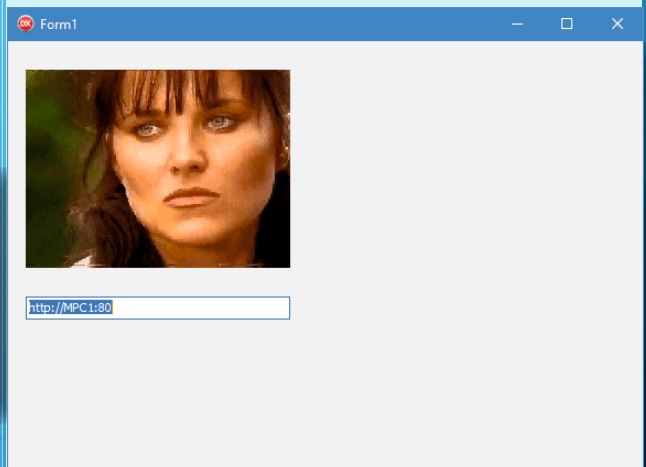


Compile and run the application.

You may see a Firewall alert since the application will try to broadcast over the network. Click "Allow access":



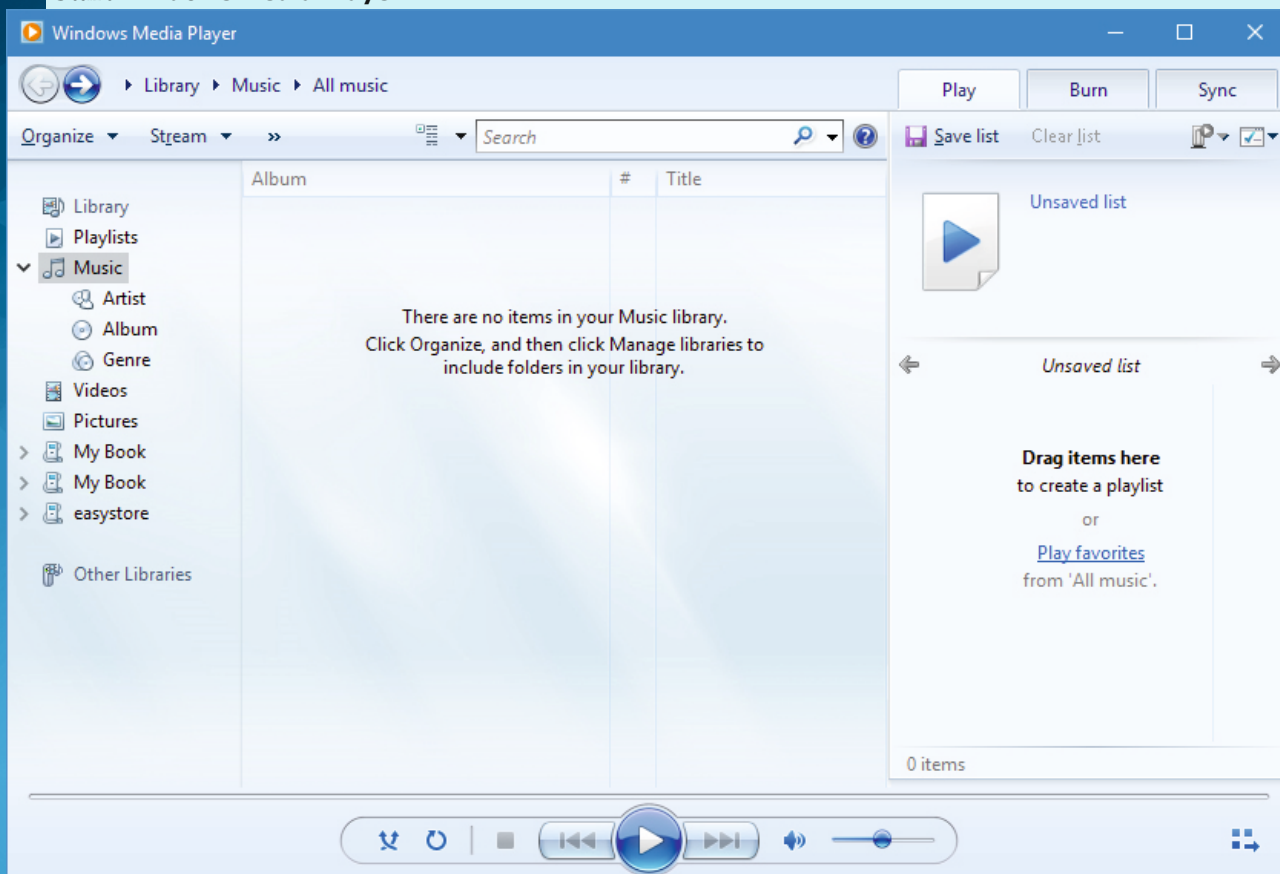
You should see the video playing in the display, and the URL in the Edit Box:



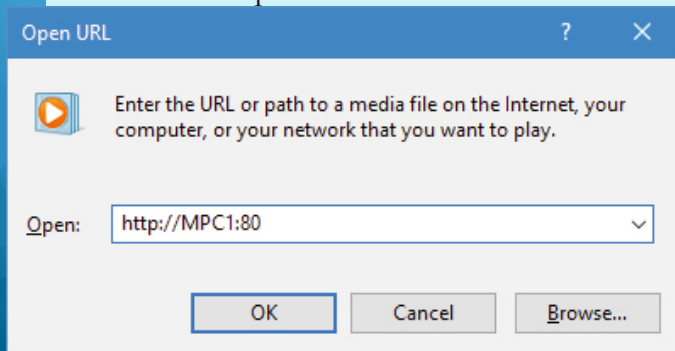
Leave the application running.

You can use the **Windows Media Player** to receive the stream.

Start **Windows Media Player**:

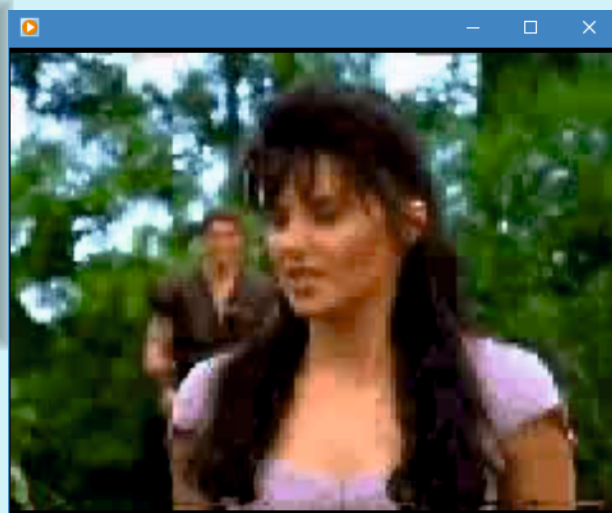


Press **Ctrl+U** to open stream with URL:



Type or Copy and paste the URL from the Edit Box of the Delphi project, and click the **"OK"** button.

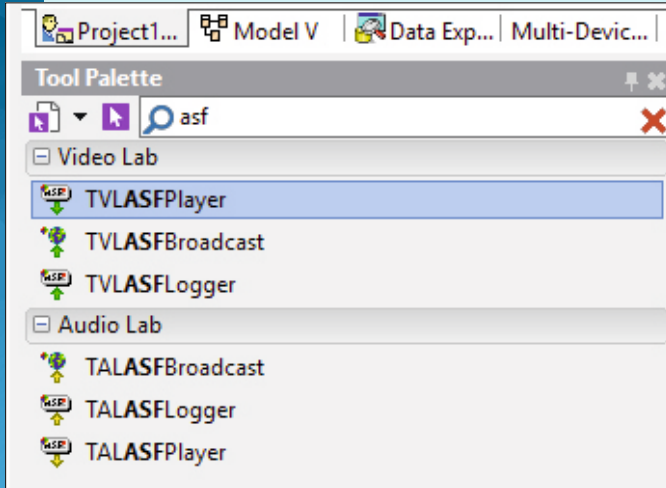
After few seconds, you will see the video from the broadcast:



We can also modify the **NDI Video receiver** project that we created earlier to receive the **ASF** stream. Reopen the **NDI Video receiver** project.

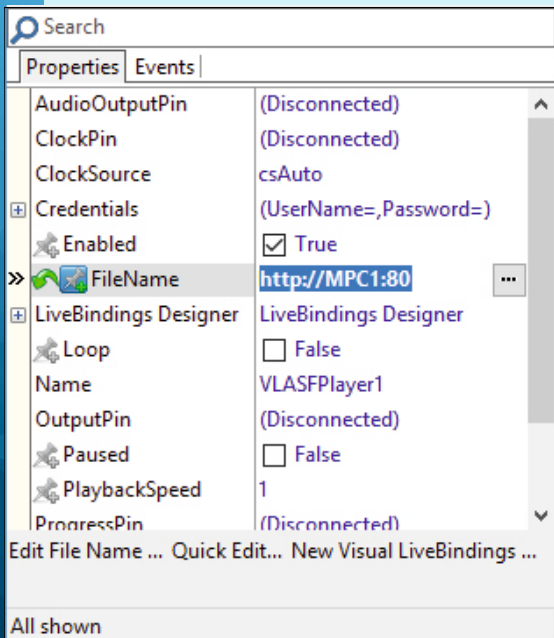
Select the **VLNDIVideoReceiver1**, and delete the component.

Type "asf" in the **Tool Palette** search box, then select **TVLASFPlayer** component from the palette:

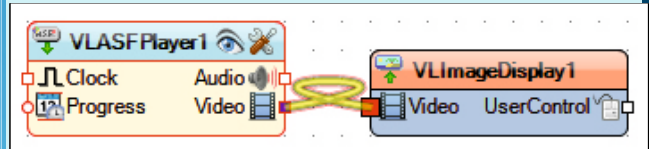


And drop it on the Form.

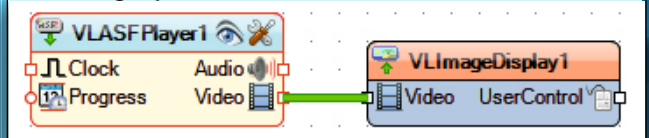
In the Object Inspector set the value of the "FileName" property to the URL from the Edit Box of the btroadcasting project:



Switch to the "OpenWire" tab. Connect the "Video" Output Pin of the **VLASFPPlayer1** to the "Video" Input Pin of the **VLImageDisplay1**:

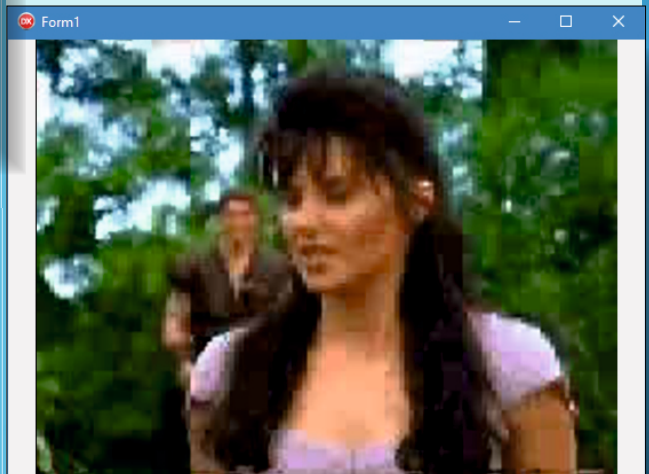


Here is the complete **OpenWire Diagram** for this project:



Compile and run the application.

After few seconds you should see the video sent from the other application on the display:



Close both applications.

In this Article, I showed you how easy it is to capture video from Directly Connected or IP Video Cameras, TV Tuners, Screen, or Online Video Streams, and how you can generate your own video from bitmaps or code. I also showed you how easy it is to broadcast video to other computers on your local network or over Internet. In the following articles, I will show you how you can mix videos together, and how to perform transition effects.



KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.06.30 BETA RELEASED!

NEW! TKBMMWISAPIRESTSERVERTRANSPORT REST CAPABLE ISAPI SERVER SIDE TRANSPORT.

- **RAD Studio 10.2 Tokyo support** including Linux support (in beta).
- **Huge number of new features** and improvements!
- **New Smart services and clients** for very easy publication of functionality and use from clients and REST aware systems without any boilerplate code.
- **New ORM OPF** (Object Relational Model Object Persistence Framework) to easy storage and retrieval of objects from/to databases.
- **New high quality random functions.**
- **New high quality pronounceable password generators.**
- **New support for YAML, BSON, Messagepack** in addition to JSON and XML.
- **New Object Notation framework which JSON, YAML, BSON and Messagepack** is directly based on, making very easy conversion between these formats and also XML which now also supports the object notation framework.
- **Lots of new object marshalling improvements**, including support for marshalling native Delphi objects to and from YAML, BSON and Messagepack in addition to JSON and XML.
- **New LogFormatter support** making it possible to customize actual logoutput format.
- **CORS support in REST/HTML services.**
- **High performance HTTPSys transport for Windows.**
- Focus on central performance improvements.
- Pre XE2 compilers no longer officially supported.
- Bug fixes
- **Multimonitor** remote desktop V5 (VCL and FMX)
- RAD Studio and Delphi XE2 to 10.2 Tokyo support
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OSX client and server support!
- **Native PHP**, Java, OCX, ANSI C, C#, Apache Flex client support!
- **High performance LZ4 and Jpeg compression**
- **Native high performance** 100% developer defined app server with support for loadbalancing and failover

Native improved XSD importer

for generating marshal able Delphi objects from XML schemas.

High speed, unified database access

(35+ supported database APIs) with connection pooling, metadata and data caching on all tiers

Multi head access to the application server, via REST/AJAX, native binary, Publish/Subscribe, SOAP, XML, RTMP from web browsers, embedded devices, linked application servers, PCs, mobile devices, Java systems and many more clients

Full FastCGI hosting support.

Host PHP/Ruby/Perl/Python applications in kbmmw!

Native AMQP support (Advanced Message Queuing Protocol) with AMQP 0.91 client side gateway support and sample.

Fully end 2 end secure brandable Remote Desktop with near REALTIME HD video, 8 monitor support, texture detection, compression and clipboard sharing.

Bundled kbmmemTable Professional

which is the fastest and most feature rich in memory table for Embarcadero products.

kbmmemTable is the fastest and most feature rich in memory table for Embarcadero products.

- Easily supports large datasets with millions of records
- Easy data streaming support
- Optional to use native SQL engine
- Supports nested transactions and undo
- Native and fast build in M/D, aggregation /grouping, range selection features
- Advanced indexing features for extreme performance

kbmmw SQL functions supported:

LOCALDATETIMEISO8601, ISO8601TOLOCALDATETIME, DATETIMEISO8601, ISO8601TODATETIME, UTCDATETIMEISO8601, ISO8601TOUTCDATETIME, PARSEUTCDATETIME, PARSELOCALDATETIME, FORMATUTCDATETIME, FORMATLOCALDATETIME

Improved support for C++ Builder

