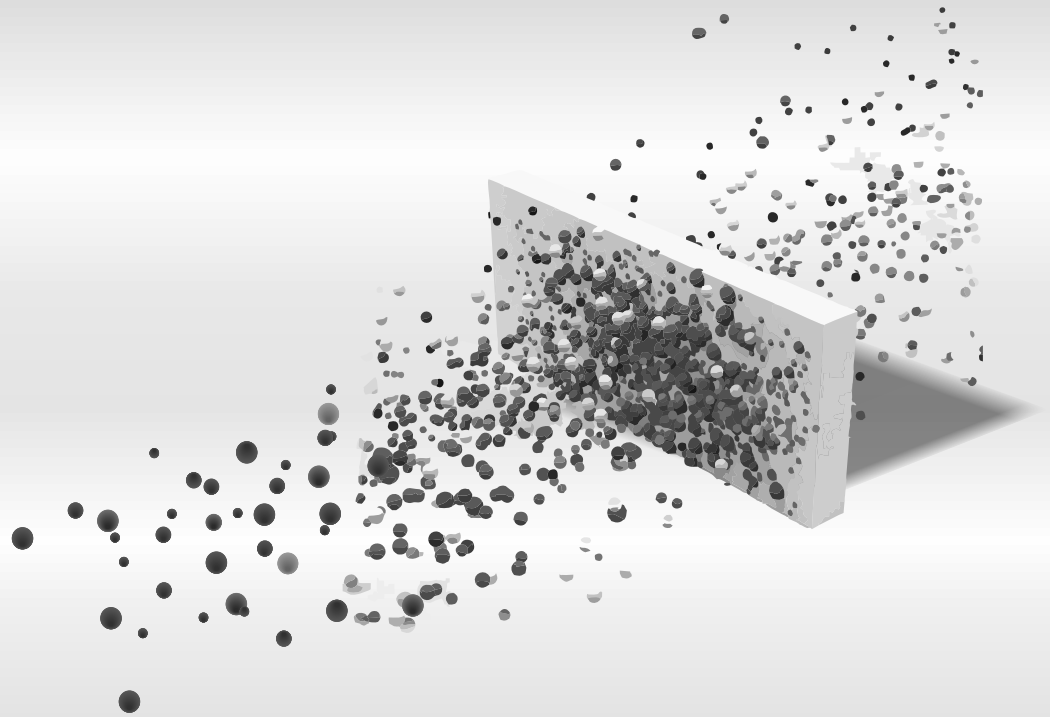


FOR DELPHI, LAZARUS, AND PASCAL  
RELATED LANGUAGES / ANDROID,  
IOS, MAC, WINDOWS & LINUX  
PRINTED, PDF, & ONLINE VIEW



# BLAISE PASCAL MAGAZINE 65



## Book reviews

### Cross platform Development with Delphi 10.2 & FireMonkey

by Harry Stahl

### Delphi in depth: FireDAC with Delphi Tokyo 10.2

by Cary Jensen

## PASCAL COIN - BLOKCHAIN

*maxbox: how to get a SHA256 or SHA256D for Blockchains*

by Max Kleiner

## Installing components in a Package

*Free Colour Buttons of high quality for Delphi and Lazarus*

by Detlef Overbeek

## Video Effects and Animations

*creating video effect without hardly any coding*

by Boian Mitov

## Futoshiki puzzle

by David Dirkse

## REST easy with kbmMW Part 3 / 4 / 5

by Kim Madsen

## FPREPORT - A new Reporting Engine

by Michael van Canneyt

## Installing Lazarus on Linux operating system and Virtual Box

*How to install Lazarus on Linux Mint in VirtualBox*

by Detlef Overbeek

## Custom Dialogs in Lazarus

by Howard page Clark

14<sup>TH</sup> OCTOBER 2017

LAST REMINDER  
YOU NEED TO REGISTER

# PASCON FOR LAZARUS

INDUSTRIEWEG 31, 3401 MA IJSSELSTEIN, NEDERLAND

## GET FREE PROGRAMS AND COMPONENTS IF YOU GO TO THE CONFERENCE

1. FreePascalReport Generator
2. kbmMW Memtable Components
3. Newest Stabel Version of Lazarus 1.8 Windows
4. Lazarus/Mint/VitualBox as VDI transportable File
5. For non subscribers: get a free subscription for Blaise Pascal Magazine for one year. (Download)
6. Free Book: "Learn to program using Lazarus (PDF)
7. Introduction New book for Lazarus "HANDBOOK LAZARUS" discount for early bird pre-orders. --> **NEW**

10.00  
Coffee  
10.30

## REGISTRATION AND RECEPTION...

### INTRODUCTION TO THE LATEST STABLE VERSION OF LAZARUS (1.8)

Mattias will show the most important new items. We will discuss the method to create your own Components and integrate in Lazarus. (*The so-called open and closed source versions and what it really means for the end user*).

**Mattias Gärtner** will show the most important change and changes of IDE features to you, we will discuss the roadmap of Lazarus, FPC and Blaise Pascal Magazine. Mattias is the main developer of Lazarus, Michael van Canneyt for FPC. Michael will be available on the conference, you will be able to put questions to the team.

Because we will show the newest version of Lazarus we also will show the consequences for viewing live on a 4k Screen, 48 inch this new version in a so called High DPI Mode. <-- NEW On Windows 10 as well Windows 7!

12.00

### RANDOM? WHAT DOES THAT REALLY MEAN? THADDY DE KONING <-- NEW

about the importance of random, real random, pseudo random and what the consequences are for your code.

- speed / secure randoms / statistics
  - issues like Delphi of Lazarus compatible random routines
  - issues like using a Raspberry Pi (or modern Intel) as hwrng. (*secure*).
- Random is still a comprehended issue...

12.45 - 14.00  
14.00

## Lunch

### KBM MEMTABLE IN CONJUNCTION WITH SQL.

kbmMW mentions a very important fact that you can use SQL in this memory table,  
- not only a very specific SQL statement but ANY kind of statement from ANY supplier:  
whether its MySQL, Oracle, Microsoft, Firebird etc.

**Something that NO OTHER MemTables offers.**

*This means you could create any kind of program without the use of a Database and still can use SQL. You can even prepare your SQL in an SQL generator and use it outcomes.*

We found **Components4Developers-owner** Kim Madsen to have the latest version available for this experiment so that we built a component group which will be available for free with closed source and if you want the source code you can of course order that.

15.00

### NEW: PASCAL RIEKENBERG WILL SHOW THE NEW FP REPORT

The first FreePascal-Report generator, a tool you will of course receive for free.  
NOT Fastreport but a **totally new developed report generator** by Michael van Canneyt.  
Imagine: create your own Report Generator and make it available for your customers...!  
The specialist that will explain it all is: Pascal Riekenberg

16.00

### ANTHONY VOGELAAR LET'S THE TIME TICK:

Coffee break

running a T-timer for lazarus and then do that job a bit more advanced running on almost nanoscale for the use of a timer on the basis of the CPU. This of course without having a graphical environment but a very precise clock.... Incredible!

**REGISTRATION BY PAYPAL** [http://www.blaisepascal.eu/agenda/lazarus\\_conference.php](http://www.blaisepascal.eu/agenda/lazarus_conference.php)  
**REGISTRATION FORM**

<http://www.blaisepascal.eu/contacts/RegistrationPasconOctober2107.php>

# BLAISE PASCAL MAGAZINE 65

DELPHI, LAZARUS, SMART MOBILE STUDIO,  
AND PASCAL RELATED LANGUAGES  
FOR ANDROID, IOS, MAC, WINDOWS & LINUX



## CONTENTS

### Book reviews

**Cross platform Development with Delphi 10.2 & FireMonkey** Page 5

by Harry Stahl

**Delphi in depth: FireDAC with Delphi Tokyo 10.2** Page 9

by Cary Jensen

**PASCAL COIN - BLOKCHAIN** Page 13

*maxbox: how to get a SHA256 or SHA256D for Blockchains*

by Max Kleiner

**Installing components in a Package** Page 20

*Free Colour Buttons of high quality for Delphi and Lazarus*

by Detlef Overbeek

**Video Effects and Animations** Page 23

*creating video effect without hardly any coding*

by Boian Mitov

**Futoshiki puzzle** Page 42

by David Dirkse

**REST easy with kbmMW Part 3 / 4 / 5** Page 47/49/52

by Kim Madsen

**FPREPORT - A new Reporting Engine** Page 56

by Michael van Canneyt

**Installing Lazarus on Linux operating system and Virtual Box** Page 61

*How to install Lazarus on Linux Mint in VirtualBox*

by Detlef Overbeek

**Custom Dialogs in Lazarus** Page 72

by Howard page Clark

Sometimes quantum particles can go through walls, as if an invisible tunnel opened up before them!

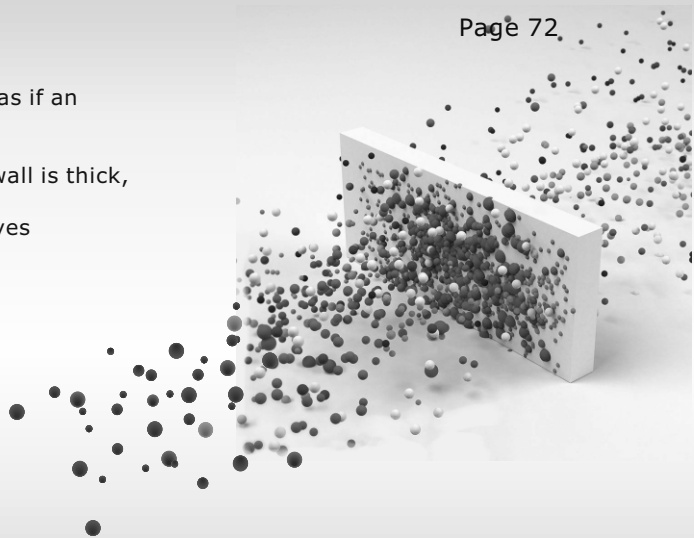
Imagine throwing an electron against a wall. If the wall is thick, the electron bounces back, which seems normal. But the electron is a quantum particle and also behaves like a wave. If the wall is very thin, the electron may be found on both sides of the wall, which means it can sometimes go through the wall.

This is called the tunnel effect, one of the main effects at the base of nanosciences.

The Image was helpfully supplied by :

<http://toutestquantique.fr/en/tunnel-effect/>

We altert the picture for deisgn -purposes.



**Important: In issue 64 there was a peace of code that was erroneus double placed in the list. Jou Now can find the right Code: HSBUTTON.zip**

### ADVERTISERS

BARNSTEN PAGE 46

COMPONENTS4DEVELOPERS PAGE 80

FASTREPORT PAGE 19

PASCON FOR LAZARUS 2017 PAGE 2

VISUINO PAGE 40/41



Publisher: Foundation for Supporting the Pascal Programming Language  
in collaboration with the Dutch Pascal User Group (Pascal Gebruikers Groep)  
© Stichting Ondersteuning Programmeertaal Pascal

<b>Stephen Ball</b> http://delphiaball.co.uk @DelphiABall	<b>Peter Bijlsma -Editor</b> peter @ blaiseascal.eu	<b>Dmitry Boyarintsev</b> dmitry.living @ gmail.com
<b>Michaël Van Canneyt,</b> michael @ freepascal.org	<b>Marco Cantù</b> www.marcocantu.com marco.cantu @ gmail.com	<b>David Dirkse</b> www.davdata.nl E-mail: David @ davdata.nl
<b>Benno Evers</b> b.evers @ everscustomtechnology.nl	<b>Bruno Fierens</b> www.tmssoftware.com bruno.fierens @ tmssoftware.com	<b>Primož Gabrijelčič</b> www.primoz @ gabrijelcic.org
<b>Mattias Gärtner</b> nc-gaertnma@netcologne.de		<b>Peter Johnson</b> http://delphidabbler.com delphidabbler@gmail.com
<b>Max Kleiner</b> www.softwareschule.ch max @ kleiner.com	<b>John Kuiper</b> john_kuiper @ kpnmail.nl	<b>Wagner R. Landgraf</b> wagner @ tmssoftware.com
<b>Kim Madsen</b> kbm @ components4developers.com	<b>Andrea Magni</b> www.andreamagni.eu andrea.magni @ gmail.com www.andreamagni.eu/wp	<b>Boian Mitov</b> mitov @ mitov.com
	<b>Paul Nauta PLM Solution Architect CyberNautics</b> paul.nauta@cybernautics.nl	<b>Jeremy North</b> jeremy.north @ gmail.com
<b>Detlef Overbeek - Editor in Chief</b> www.blaiseascal.eu editor @ blaiseascal.eu	<b>Howard Page Clark</b> hdpc @ talktalk.net	<b>Heiko Rempel</b> info@rompelsoft.de
<b>Wim Van Ingen Schenau -Editor</b> wisone @ xs4all.nl	<b>Peter van der Sman</b> sman @ prisman.nl	<b>Rik Smit</b> rik @ blaiseascal.eu www.romplesoft.de
<b>Bob Swart</b> www.eBob42.com Bob @ eBob42.com	<b>B.J. Rao</b> contact@intricad.com	<b>Daniele Teti</b> www.danieleteti.it d.teti @ bittime.it
<b>Anton Vogelaar</b> ajv @ vogelaar-electronics.com	<b>Siegfried Zuhr</b> 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@blaiseascal.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 )

	Dutch	Shipment in Netherl	Internat. excl. VAT	Internat. incl. VAT	Shipment
<b>Printed Normal Issue 44 pages</b>	€ 90	No Extra	€ 90	€ 95,40	€ 60
<b>Printed Extended Issue 80 pages</b>	€ 150	€ 25	€ 150	€ 159	€ 80
<b>Electronic Extended Issue 80 pages</b>	€ 60,50	—	€ 50	€ 60,50	—

**Printed magazine edition**

10 issues per annum, 44-page Delphi-only section: € 90.-- This includes postage, VAT at 6 % and all code and programs accompanying the articles.

Excluding postage the 44-page edition is € 60.-- per annum.

10 issues per annum 80-page Delphi + Lazarus sections: € 150 plus € 80 for postage.

**Digital magazine edition (PDF format)**

10 issues per annum 80-page Delphi + Lazarus sections: € 50.-- (excluding VAT at 21%).

For the months to the end of 2017 we are trialling a fuller 80-page edition of the magazine, with two sections, the first with a Delphi focus, and the second with a Lazarus/FPC focus.

We will decide, based on reader feedback, at the end of 2017 whether to continue with this larger format magazine, or revert to the 44-page format you know from recent issues.

Subscriptions can be taken out online at [www.blaiseascal.eu](http://www.blaiseascal.eu) or by written order, or by sending an email to [office@blaiseascal.eu](mailto:office@blaiseascal.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: Paypal

Name: Pro Pascal Foundation-Foundation for Supporting the Pascal Programming Language (Stichting Ondersteuning Programeertaal 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 / Tel.: + 31 (0) 30 890.66.44 / Mobile: + 31 (0) 6 21.23.62.68

[office@blaiseascal.eu](mailto:office@blaiseascal.eu)

**Copyright notice**

All material published in Blaise Pascal is copyright © SOPP Stichting Ondersteuning Programeertaal 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.

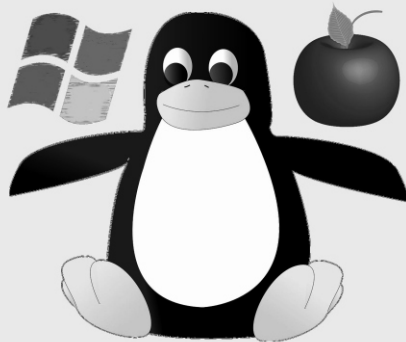
Member and donor of



WIKIPEDIA

# Cross-Platform Development

with Delphi 10.2 & FireMonkey



for Windows, MAC OS X (macOS)  
&  
Linux

Harry Stahl

Title:  
**CROSS-PLATFORM-DEVELOPMENT  
WITH DELPHI TOKYO 10.2**

Author:  
Harry Stahl  
Publisher:  
Harry Stahl  
City: Bonn, Germany  
Copyright (2017),  
All rights reserved  
**ISBN: 9781549545764**  
Imprint:  
Independently published



The book is sold in two versions:  
<https://www.amazon.de/dp/1521136661>  
**Price of the English Version: € 49.90**  
**Price of the Ebook version € 29,90**

Selling an **Ebook** does make sense but a PDFversion would also be a wonderful option. A printed book however is in practice much more useful for obvious reasons.

I personally like to have them both: the electronic version for rapid topic location using the index, while during development a printed book is like an extra screen and adds to the fun of programming and problem solving

## WHY BUY THIS BOOK:

This book is written for experienced **VCL-developers**, so will be rather laborious for beginners who would like an easy start with **FireMonkey**, or for those already working with **FireMonkey** and searching for solutions.

Experienced users who previously developed for Windows have usually - if migrating to FireMonkey - questions that seem to be hard to find answers for.

First of all they need to find the solution for common problems like connecting with Windows PC, MAC or Linux and their various setting dialogs.

Often there are only small differences between FireMonkey- and VCL components, differences which sometimes lead to failure during development.

Finding the differences is often costly. This book explains the minor and major differences between the well-known VCL components.

The use of FireMonkey components makes more sense if they are used for cross platform development.

Having noticed this, Windows and e.g. MAC handle a lot of functions completely differently. For example: passing parameters at the start of the program.

This book provides a great many answers you might need and therefore saves a lot of time. If the MAC or Linux environment is new to you or you have no basic information about handling of files, available storage locations or required developer tools, you will find at least the most important information in this book. A new chapter in this book is 3D-programming. Here you will find the basic principles needed to understand them and some easy to do sample-applications.

Please note: this book does not handle the topic "databases".

This is due to the fact that the author does not use Delphi database components, but instead uses his own solution for working with "databases".

## Conclusion:

**its good to have a book like this and it was worth waiting for the English version, which most people find easier to understand. If so asked, I would advise you to buy it. It's worth it....**

## TABLE OF CONTENTS

Foreword	8
Introduction	9
About the book	9
About the author	9
Contact information	9
<b>Chapter 1: What is FireMonkey? 10</b>	
<b>Chapter 2: How to use the FireMonkey components 11</b>	
Section 1: Getting Started	11
Section 2: New FireMonkey project	13
FireMonkey desktop application (Multi Device Application)	13
Using the Multi Device Designer (Fire UI)	15
Form inheritance with the Multi Device Designer	18
Reverting to inherited settings	19
Creating a platform-specific event handler with the Multi-Device Designer	20
Section 3: A first FMX-program (analog clock)	21
Section 4: Selected FireMonkey components	27
TButton (with Trimming)	27
TEdit (without PasswordChar)	27
TExpander	28
TForm (furthermore with caption)	28
TFrame	28
TPanel	28
TRectangle	29
TCheckBox, TRadioButton (IsChecked)	29
TGroupBox with TRadioButtons	29
Tswitch	29
TImage	30
TImageControl	31
TImageViewer	31
TImageViewer (to use with Livebindings Designer)	31
TLabel (new property FontColor)	35
TPathLabel	35
TPath	35
TImageList (not available - but compensation possible)	36
TListBox (no TCheckListBox, but ShowCheckboxes)	39
All Components (except the form)	49
Several Components (Properties with additional type-qualifying)	49
TMenuItem (without ImageIndex)	49
TMainMenu (Handling MAC Menus)	50
TMemo (CaretPosition, no Modified, FindNext-replacement)	51
TDropTarget (how drag & drop works in FireMonkey)	58
TRichEdit (not available - but possible for replacement via 3rd-party)	60
TPageControl (Not available - but replacement available)	60
TStringGrid (works different)	60
TGrid (Image and other elements in the Grid)	65
TStringGrid-alternative (TMSFMXGrid)	66
THeader (not sections, but items)	66
THeaderControl (is not available under FireMonkey)	66
TProgressBar (not "position" but value)	66
TTabControl (no Ownerdraw)	67
TTrackbar (helpful property "tracking")	67
TSpeedButton (without Bitmap)	67
TStatusBar (a way to compensate the missing "Panels")	67
MessageDlg (e.g. not directly usable with mtWarning)	67
Section 5: The FireMonkey Style-Designer	68
a) Using the Styles Editor	68
b) Styles in FireMonkey - an overview	71
c) How to convert VCL Styles to FireMonkey Styles	74
d) Using FireMonkey Styles	75
e) Understanding FireMonkey-Styles	76
<b>Chapter 3: Tips and tricks for Cross-Platform Development 79</b>	
Section 1: Starting other programs	79
Section 2: Get the program directory and program data directory	80
Section 3: Catch to the program passed start-up parameters	83
Section 4: "Hello World" - Multilingual programs and new markets	87
Section 5: Apply sandboxing and Entitlements properly	91
Section 6: Using MAC APIs (POSIX, CORE and Cocoa) in Delphi	95
<b>Chapter 4: Requirements for Cross-Platform Development 100</b>	
Section 1: Setting up Windows PC and MAC PC	100
Section 2: Enabling MAC OSX Platform	103
Section 3: Provisioning and deployment (MAC)	107
1. Submission to the APPLE App Store	108
2. How to create a .dmg file for distribution outside the Apple App Store	111
3. How to create your own setup package with the Application Developer ID / Installer	113
a) Hot to request a Developer ID certificate and an Application Developer Installer ID	114
b) Working with the code-signing tool and Package Maker	116

## Chapter 4: Requirements for Cross-Platform Development 100

Section 1: Setting up Windows PC and MAC PC	100
Section 2: Enabling MAC OSX Platform	103
Section 3: Provisioning and deployment (MAC)	107
1. Submission to the APPLE App Store	108
2. How to create a .dmg file for distribution outside the Apple App Store	111
3. How to create your own setup package with the Application Developer ID / Installer	113
a) Hot to request a Developer ID certificate and an Application Developer Installer ID	114
b) Working with the code-signing tool and Package Maker	116

## Chapter 5: Cross-Platform development with Linux 120

Section 1: Setting up Windows and Linux-PC	121
Section 2: Enabling the Linux-Platform	125
Section 3: Provisioning and deployment (Linux)	127
Section 4: A first Linux-Console-Application	128
Section 5: Linux-Server-Console-Application and Client-Application	130
Section 6: Create an Application as Service (Daemon)	137
Section 7: Delphi units for Linux	139

## Chapter 6: Working with Graphics in FireMonkey 142

1. FireMonkey TBitmap versus Windows TBitmap	142
2. TBitmapData instead of ScanLine for bitmap manipulation	142
3. How to change the alpha channel of a TBitmap	143
4. How to draw on the canvas of a bitmap	144
5. How to turn graphics, flip, invert or color to gray	145
6. How to draw a bitmap scaled	148

## Chapter 7: 3D-Programming 149

Section 1: Overview	149
1. 3D-Objects	149
2. Cameras	149
3. Screen Projections	149
4. Rotations	150
5. Light	150
6. Materials	152
Section 2: The 3D Coordinate System	154
Section 3: 3D-Application "Atomic Model"	158
Section 4: 3D-Application "Solar Model"	163

## Chapter 8: Animations, Transitions and Effects 166

## Chapter 9: Sending and receiving messages with the TMessageManager 170

Section 1: Simple Messaging-Demo	171
Section 2: Enhanced Messaging-Demo	173

## Chapter 10: Useful third-party components for FireMonkey 177

1. TMS-Components	177
2. Report-generator: FastReport FMX	178
3. RemObjects-Application Framework (Hydra)	179
4. Other components	179

## Chapter 11 How to - tips & tricks for FMX 180

R1 ... Get the display resolution?	180
R2 ... Check whether the Escape, Ctrl or Alt key is pressed	180
R3 ... Use folder names under Windows and MAC properly	182
R4 ... Use search-mask for "all files" in Windows and MAC OS X properly	184
R5 ... Avoid looping symlink folders (Alias)	184
R6 ... In which situations file symlinks functions play a role otherwise	185
R7 ... Determine the control under the mouse position	186
R8 ... Find out on which MAC OS X operating system the program is running	186
R9 ... Get the current user name in Mac OS X / Linux / Windows	188
R10 ... Send files as an attachment of an e-mail with the system mail program	188
R11 ... Provide the user with help-files under Win & MAC	190
R12 ... Drag and drop text from external source (eg browser) to a TEdit box	192
R13 ... Store additional information in standard objects	193
R14 ... Using ActiveControl	193
R15 ... Replace OnDrawItem event of the ListBox from VCL with the OnPainting event of the TListBoxItems	194
R16 ... Load Bitmap from resource file (for retina display)	195
R17 ... Swap items in a listbox	197
R18 ... Swap items in a Listbox via Drag & Drop	198
R19 ... Using FMX functions in a VCL application via DLL	198
R20 ... Draw text in TGrid right or centered	204
R21 ... Draw text in TStringGrid right or centered	206
R22 ... Working with the "visible" property of controls	207
R23 ... Prevent unintended shortening of TLabel text	207
R24 ... How to show a pop-up menu at a special position	208
R25 ... Determine the document directory	209
R26 ... Improve the font quality (especially on Windows)	209
R27 ... Select a folder with a dialog	209
R28 ... Let a column in a string grid occupy the remaining space	210
R29 ... Create missing components with Frames	211
R30 ... Moving controls at runtime in the form	215

## Chapter 12: Outlook 218

Index 219

## CHAPTER 1: WHAT IS FIREMONKEY?

FireMonkey, usually abbreviated as "FMX", is a software component library or vector-based framework, which allows cross-platform application development for Windows, MAC OS X (or "macOS"), Linux, iOS and Android, often with the same source code. The first FMX version was released with XE2, with XE3 followed an extended FMX version, which was often called "FireMonkey 2".

Since then FMX has been heavily reworked with every Delphi version, so the developer often had to make a number of adjustments when switching to the latest FMX version.

Fortunately the functionality of FMX increased with every new Delphi-version, so that today we have a very powerful framework, with which you can do not only everything that is possible with the VCL, but also much more. All components are freely rotatable and individually scalable. There are also a number of 3D components that can be used to write 3D programs. Finally, the effects and animations are to be mentioned, which give FMX another unique feature. The representation of the components is supported by the GPU (Graphic Processing Unit), which makes the output faster and more fluid.

Under Windows, the GPU is addressed with DirectX, under Mac with OpenGL and under iOS / Android with OpenGL / ES.

### History

FireMonkey was originally developed by Eugene Kryukov (company KSDEV, Uland-UDE in Russia). The product was known as VGScene. In 2011 the framework was purchased by Embarcadero and integrated in Delphi XE2 as a new framework, in addition to VCL. From XE3 it is only from the enterprise version on an integral part of Delphi, for the professional version you have to purchase it separately as a so-called mobile pack. Since Delphi 10 Berlin you can create 64-bit applications for Windows and also for IOS, for Mac and Android it remains so far with the 32-bit version. Starting with Delphi 10.2 Tokyo, the Linux platform (64-bit) is also supported, but only for the creation of console applications.

## Outlook

In relation to the VCL platform, the main innovations and enhancements are found now at FMX. There are always new components and features added to the components. In this respect I see here the future of software development with Delphi.

Since May 2017, with the "FMXLinux" Add-on, we have also a possibility to develop Linux applications for the desktop with Delphi (more info on fmxf Linux: <http://www.fmxf Linux.com>).

So, do not be surprised if you already see some screenshots of Linux desktop programs here in the book. These were created with Delphi and the FMXLinux Add-on and look just better, as result displays in console windows.

Cross-Platform-Development with Delphi Tokyo 10.2

### Form inheritance with the Multi Device Designer

Also, it is a space-saving method because Delphi creates for each view a separate form and only this is included for the correspondent platform. When the files are created, in a file manager it looks like the following:

Name	Typ	Datum	v	Größe
FMandant.Macintosh.fmx	fmx	Di 30.09.2014 23:42:08		1 kb
FMandant.fmx	fmx	Di 30.09.2014 23:42:08		9 kb
FMandant.pas	.pas	Di 30.09.2014 23:41:30		8 kb

FMandant.fmx is the master form file. This file serves as a master for the respective generated platform. If we have only this master form and want to create a program for the MAC OS X, we can use this master form only.

Here we want to create a Windows program and also a version for the MAC OS X. Therefore we have chosen here "OS X Desktop" in the right drop-down list so that Delphi will create a form for this platform. This is the file "FMandant.Macintosh.fmx".

The specific platform forms work on the principle of form inheritance (similar to the "TFrame"). When we open the MAC form in a text editor, it looks like this:

```

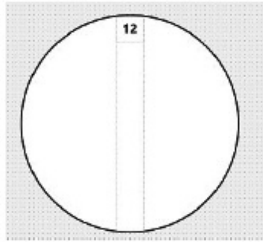
inherited F_Mandant_Macintosh: TF_Mandant_Macintosh
  DesignerMasterStyle = 0
  inherited Panel1: TRectangle
    inherited pcl: TTabControl
      inherited TabSheet1: TTabItem
        Size.Width = 68.000000000000000000
        inherited Label1: TLabel
          Size.Width = 99.000000000000000000
          Size.Height = 18.000000000000000000
        end
        inherited btnOK: TButton
          Position.X = 436.000000000000000000
          inherited Image2: TImage
            Visible = False
          end
        end
      end
    end
  end
end

```

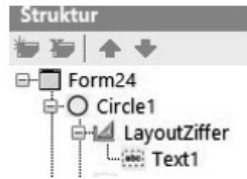
While the form master-file consists of 250 lines, the derived MAC form-file has only 42 lines. The reason is that only the changes in relation to the recognized master form will be saved here. The button is now on the right side instead of the left. Therefore it has a different

## Cross-Platform-Development with Delphi Tokyo 10.2

The preliminary result looks like this:



The structure list looks like this:



If it looks different on your PC, you can move the elements by drag and drop to the right position.

We could go on and copy this TLayout ("LayoutZiffer") 11 times and increase the value for RotationAngle each time by 30 degrees and reduce the value for "RotationAngle" for the TText-Element by 30 degrees (like I have demonstrated it in the mentioned video above).

But instead of this we reach th

5. Place the following text into th

```

procedure TForm24.FormCreate
var
    L: Integer; LA : TLayout;
begin
    for L := 1 to 11 do begin

        // create a copy of the
        LA := TLayout (circle1
        ('clocknumber', true
        if LA <> NIL then begin
            LA.Parent := circle1;
            LA.RotationAngle := 0;

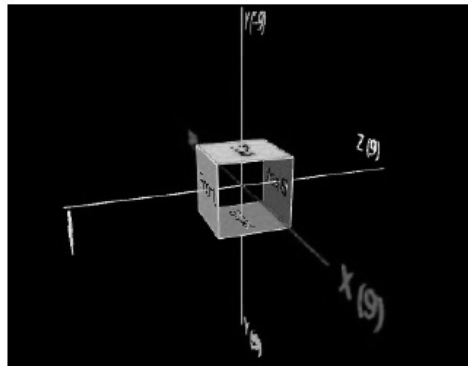
            // We have only one
            T := TText (LA.Children[0]);
            if T <> NIL then begin
                T.Text := L.ToString;
                T.RotationAngle := 0;
            end;
        end;
    end;
end;
    
```

## Cross-Platform-Development with Delphi Tokyo 10.2

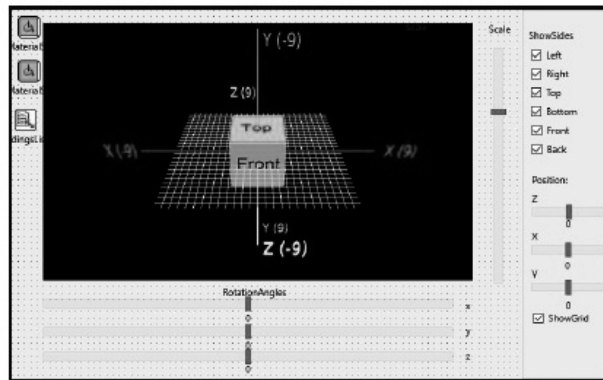
With a value of 0,0 for "RotationCenter" the rotation would have its center point at the top left of the component.

For 3D objects, the value is set initial to 0,0,0. You can not change this at the design time, only at runtime (!).

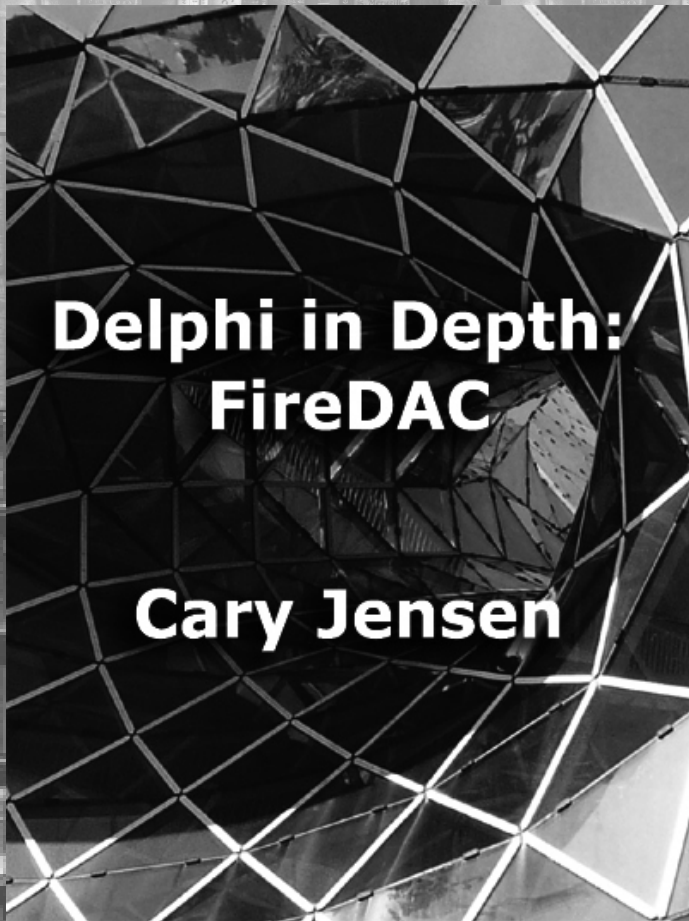
In the following picture you can clearly see the center of rotation exactly in the center of the 3D object:



Here it is also worthwhile to look at the corresponding demo program, that I have developed, in order to see the different points in the 3D space. At the design time the program looks like this:







**Title:**

**DELPHI IN DEPTH: FireDAC WITH DELPHI TOKYO 10.2**

**Author:** Cary Jensen

**Project Editor:** Loy Anderson

**Contributing Technical Editors:**

Dmitry Arefiev, Holger Flick, Jens Fudge, and Bruce McGee

**Cover Designer:** Loy Anderson

**Indexer:** Cary Jensen

**ISBN-10: 1546391274**

**ISBN-13: 978-1546391272**

**ISBN-10: (e-book edition)**

**Published by**

Jensen Data Systems, Inc., USA.

<http://www.JensenDataSystems.com/firedacbook>

Publish date: May 11, 2017.

Paperback: 558 pages

Language: English

**LINKS FOR PURCHASING THIS BOOK:**

Ebook version from **FastSpring:**

Retail price for ebook: \$44.99 USD

Buy printed book at **CreateSpace**

(CreateSpace is Amazon's publishing company)

Amazon US: Buy Book USA

Amazon Canada: Buy Book Canada

Amazon.co.uk: Buy Book Amazon.co.uk (UK)

Amazon.de Germany, Switzerland, Austria:

Buy Book Amazon.de Amazon.fr France:

Buy Book Amazon.fr (France) Amazon.es Spain:

Buy Book Amazon.es (Spain and other counties)

Amazon.it Italy:

Buy Book Amazon.it (Italy) Amazon.jp Japan: Available soon

If the book is not yet available from Amazon in your country, you can buy this book directly from CreateSpace (Amazon's publishing company), and have it shipped to your address.

Retail price for printed book:

\$49.99 USD

€49.00 EURO

£40.00 GBP

This book covers the current version of Delphi, Delphi 10.2 Tokyo as well as previous versions of Delphi. There will also be an accompanying download with source code which you can download from this page.

**CHAPTER TITLES**

Chapter Titles	v
Table of Contents	vii
About the Author	xvii
About the Technical Reviewers	xix
Acknowledgements	xxi
Introduction	1
Chapter 1 Overview of FireDAC	5
Chapter 2 Connecting to Data	15
Chapter 3 Configuring FireDAC	47
Chapter 4 Basic Data Access	81
Chapter 5 More Data Access	109
Chapter 6 Navigating and Editing Data	145
Chapter 7 Creating Indexes	165
Chapter 8 Searching Data	197
Chapter 9 Filtering Data	217
Chapter 10 Creating and Using Virtual Fields	259
Chapter 11 Persisting Data	297
Chapter 12 Understanding FDMemTables	329
Chapter 13 More FDMemTables:	
Cloned Cursors and Nested DataSets	369
Chapter 14 The SQL Command Preprocessor	397
Chapter 15 Array DML	425
Chapter 16 Using Cached Updates	439
Chapter 17 Understanding Local SQL	487
Appendix A Code Download, Database Preparation, and Errata	507
Index	519

**Conclusion about the book:**

The book is as always a very detailed and in-depth look at FireDAC as you would expect from Cary Jensen.

I must say it was absolutely necessary someone would help us understand FireDAC and its possibilities.

It is very useful because there are lots and lots of subjects very hard to come by if not explained.

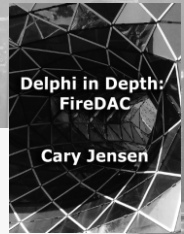
It will help to make it much easier to work with Databases and even make you work faster.

I consider it to be a very helpful book and if you have a look at all the items it covers and shows: you will learn a lot of new details about the subjects.

Great.

It was about time we would get such a helpful book.

**It's a must have.**



## ABOUT THE TECHNICAL REVIEWERS

### Dmitry Arefiev

Dmitry Arefiev is the creator of AnyDAC, the product that eventually became FireDAC. He is currently the FireDAC architect for Embarcadero Technologies, the makers of RAD Studio and Delphi.  
Email: [darefiev@gmail.com](mailto:darefiev@gmail.com)

### HOLGER FLICK

Dr. Holger Flick is a well-known member of the Delphi Community, and has worked with Delphi and Borland Pascal before Delphi. While achieving a Degree in Computer Science and a Doctorate of Engineering, he was part of several developer teams at Borland and later CodeGear. This gave him the means to gain first-hand knowledge of the tools and frameworks. He wrote several articles and spoke at many Delphi Road Shows, seminars, and conferences. When developing software with Delphi, his focus is on database-driven applications for both desktop and mobile platforms. Since 2016, Holger heads his new brand "Flix Engineering" and is available for training, development, and consulting services.

URL:

<https://flixengineering.com/>

blogTwitter:

<https://twitter.com/hflickster>

LinkedIn:

<https://de.linkedin.com/in/hflick>

Email: [info@flixengineering.com](mailto:info@flixengineering.com)

### Jens Fudge

Jens Fudge has been working with Delphi since 1995, when it first came out. He has built mainly database systems for a lot of various customers in different areas like railroad companies, airports, cement factories and even a government application. Jens is an Embarcadero Delphi MVP, and works as a trainer and consultant for many different companies, and is also a frequent speaker at international and national conferences. Apart from being a Delphi developer and consultant, Jens also brews beer, wine and mead, and shoots archery. The latter has inspired Jens to the name of his company, which is Archersoft. Jens won the Gold medal in archery at the Paralympic Games in Barcelona, Spain in 1992.

Email: [Jens.fudge@archersoft.dk](mailto:Jens.fudge@archersoft.dk)

### Bruce McGee

Bruce McGee operates a software consulting company named Glooscap Software in Toronto, Ontario, Canada. He has been a user of and advocate for Delphi for many years, and continues to work with it daily. He is also a big fan of continuous learning, especially in the software development field, and the need to constantly hone our craft.

Blog: <http://www.glooscap.com/>

## WHO IS THIS BOOK FOR

This book is intended for the Delphi database developer. In it you will find information at nearly every level of application development. If you are new to database development in Delphi, you will find basic information about how the TDataSet interface works. For example, how to navigate records, the concept of the current record, accessing fields, and how to edit data.

If you are an advanced database developer, you too will find valuable information. For example, how to define dynamic master-detail relationships, the convenience of nested datasets, and the power of cached updates. In order to use the examples found in this book you will need to be using Delphi XE6 or later, and ideally Delphi 10 Seattle or later. At a minimum, you will need the profession version of these products, and will also need to install either the InterBase server or the InterBase developer edition. There are a few more requirements, and you will find out more about these in Appendix A, which you should read before continuing to Chapter 1:

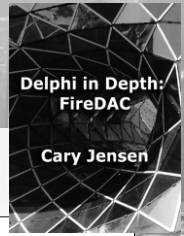
## Overview of FireDAC. Conventions

Most of the examples in this book make use of FDQuery components, which are used to execute SQL (Structured Query Language) statements. In this book, I am pronouncing SQL as "es"- "que"- "el," and not "sequel." What this means is that I will say "an SQL statement," instead of "a SQL statement." Another convention that I use is to drop the T in most references to a class. For example, while I will occasionally speak strictly about a class, say TFDQuery, I will most often refer to instances of this class as FDQueries, and then more conversationally as "queries." My main goal is readability. To me, the constant use of the T in a class name makes the text harder to read.

Another convention relates to the sample projects that accompany this book. In almost every case, when I show a code segment, it is code that can be found in a sample project from the code download. The first time I refer to a given project in a chapter, I include a note indicating the name of the project as it appears in the code download. I do not repeat this note in subsequent references to that project in the same chapter.

Another convention concerns how screenshots are referenced. This book includes both figures and illustrations. All figures are numbered, and include a caption. Illustrations are not numbered, and do not include captions. Illustrations are used for small screenshots that are discussed in the text that immediately precedes the screenshot.

By comparison, figures may not appear on the same page from which they are referenced, and may be referred to again later in the chapter. It's a minor point, but one that I want to make in case you start wondering why some screenshots lack a caption. There is one last thing. This book is about techniques involving FireDAC. And while FireDAC itself is cross-platform, almost every one of the sample projects is a VCL (Visual Component Library) example that runs only on Windows. Since Delphi is a Windows-based IDE, it is guaranteed that every reader of this book will be running Windows. Writing FireMonkey applications for iOS, Android, or OSX (Mac) involves additional technologies, and I didn't want to get bogged down with discussions of LiveBindings (which I do cover), the platform assistant, and FireMonkey component configuration. I know that some readers will be unhappy about this decision, but I wanted you to know that I had my reasons.



# Table of Contents

Dedication ..... 3

Chapter Titles ..... v

Table of Contents ..... vii

About the Author ..... xvii

    Cary Jensen ..... xvii

About the Technical Reviewers ..... xix

    Dmitry Arefiev ..... xix

    Holger Flick ..... xix

    Jens Fudge ..... xx

    Bruce McGee ..... xx

Acknowledgements ..... xxi

Introduction ..... 1

    Who Is This Book For ..... 2

    Conventions ..... 2

Chapter 1 Overview of FireDAC ..... 5

    FireDAC Features ..... 6

*Cross-Platform Support* ..... 7

*Exceptional Support for Databases* ..... 7

*Flexible Queries Using the SQL Command Preprocessor* ..... 8

*Blazing Performance with Array DML* ..... 8

*Support for a Variety of Query Execution Modes* ..... 9

*Powerful Monitoring Capabilities* ..... 9

*Cached Updates* ..... 10

*Result Set Persistence* ..... 10

*Data Type Mapping* ..... 11

*Local SQL* ..... 11

*Additional Features* ..... 12

        Connection Recovery ..... 12

        Advanced Transaction Support ..... 12

        Built-In Dialog Support ..... 12

        Support for Database-Specific Services ..... 12

        Customizable Data Access ..... 13

        Batch Move Support ..... 13

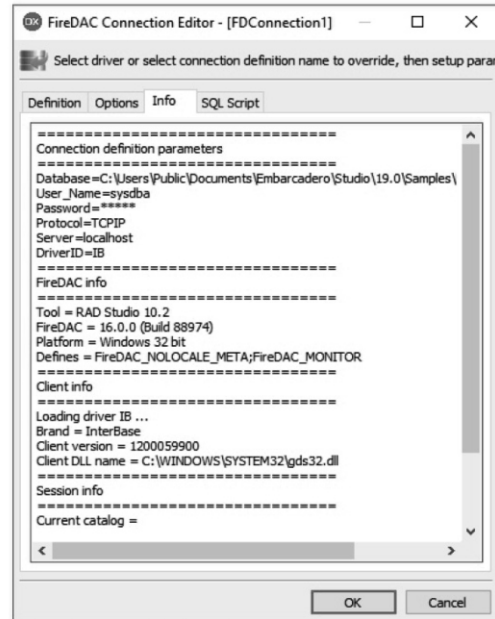


Figure 2-3: The Info tab of the FireDAC Connection Editor

Finally, the SQL Script tab permits you to enter SQL statements that you want to execute ad hoc against this connection. This can be very useful if you want to perform a quick operation such as viewing some data or creating a new table. When you are done setting your connection parameters and options, close the Connection Editor by clicking the OK button. We are now ready to finish this simple example.

## 22 Delphi in Depth: FireDAC

7. Add an FDQuery to your data module. (If you have an FDConnection component on the module to which you add an FDQuery, the query sets its Connection property automatically. If that does not happen, set the FDQuery's Connection property to FDConnection1 before continuing.)
8. Right click this FDQuery and select FireDAC Query Editor..., or simply double-click on the FDQuery. Delphi will respond by displaying the Query Editor, shown in Figure 2-4.

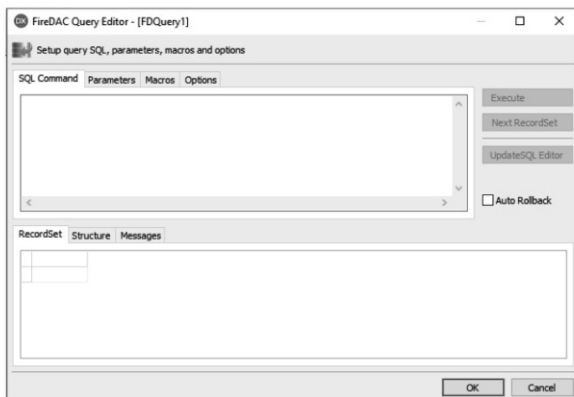
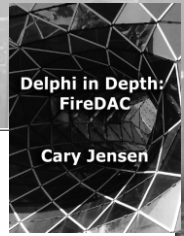


Figure 2-4: The SQL Command tab of the FireDAC Query Editor

9. In the SQL Command pane (the upper part) enter the following query:

```
SELECT * FROM Customer
```

You can now test the query, if you like, by clicking the Execute button. When you are done, click OK to return to the form. Clicking OK saves the query you entered into the FireDAC query's SQL property. If you instead click Cancel, the query text is lost.



## 38 Delphi in Depth: FireDAC

```
Database=C:\Users\Public\Documents\Embarcadero\
Studio\18.0\Samples\Data\dbdemos.mdb
```

Each named connection appears as a section name in the ini file, and the name/value pairs in this section correspond to the same name/value pairs that would otherwise appear in the FDConnection.Params property following the definition of a temporary connection.

If you want to use a connection definition file using a name other than FDConnectionDefs.ini, or in a location other than the application's current directory, you simply copy the section or sections of the default ini file and paste them into your custom ini file. In addition, you either set the ConnectionDefFileName property of a manually placed FDManager component, or you set the ConnectionDefFileName property of the automatically created FDManager at runtime, prior to attempting to connect any of your FDConnection components.

Assuming that we have copied the NewMSAccess section from the default connection definition file, and pasted it into a file named conn.ini in the application's working directory, the following runtime code will configure the automatically created FDManager prior to connecting an FDConnection:

```
procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  FDManager.ConnectionDefFileName :=
    ExtractFilePath(ParamStr(0)) + 'conn.ini';
  //This next line is not necessary if the ConnectionName
  //property of FDConnection1 was set at design time.
  FDConnection1.ConnectionName := 'NewMSAccess';
  FDConnection1.Open;
end;
```

*Note: We could have set the FDConnection's ConnectionName property at design time as well, since this is a published property of the FDConnection class.*

Alternatively, if you have manually placed an FDManager component on your form, and set its ConnectionDefFileName property to conn.ini (this is a relative path, defaulting to the application's executable directory), your application will automatically use the named connection in the connection definition file at runtime. The following figure shows the Object Inspector where a manually

## 38 Delphi in Depth: FireDAC

```
Database=C:\Users\Public\Documents\Embarcadero\
Studio\18.0\Samples\Data\dbdemos.mdb
```

Each named connection appears as a section name in the ini file, and the name/value pairs in this section correspond to the same name/value pairs that would otherwise appear in the FDConnection.Params property following the definition of a temporary connection.

If you want to use a connection definition file using a name other than FDConnectionDefs.ini, or in a location other than the application's current directory, you simply copy the section or sections of the default ini file and paste them into your custom ini file. In addition, you either set the ConnectionDefFileName property of a manually placed FDManager component, or you set the ConnectionDefFileName property of the automatically created FDManager at runtime, prior to attempting to connect any of your FDConnection components.

Assuming that we have copied the NewMSAccess section from the default connection definition file, and pasted it into a file named conn.ini in the application's working directory, the following runtime code will configure the automatically created FDManager prior to connecting an FDConnection:

```
procedure TDataModule1.DataModuleCreate(Sender: TObject);
begin
  FDManager.ConnectionDefFileName :=
    ExtractFilePath(ParamStr(0)) + 'conn.ini';
  //This next line is not necessary if the ConnectionName
  //property of FDConnection1 was set at design time.
  FDConnection1.ConnectionName := 'NewMSAccess';
  FDConnection1.Open;
end;
```

*Note: We could have set the FDConnection's ConnectionName property at design time as well, since this is a published property of the FDConnection class.*

Alternatively, if you have manually placed an FDManager component on your form, and set its ConnectionDefFileName property to conn.ini (this is a relative path, defaulting to the application's executable directory), your application will automatically use the named connection in the connection definition file at runtime. The following figure shows the Object Inspector where a manually

### GENERAL FETCHING

These properties affect how records are retrieved from the underlying database.

Property	Description
AutoClose	When True, the dataset's cursor is closed after fetching records. Default is True. Set AutoClose to False when an SQL command produces several cursors.
AutoFetchAll	Defines which records are retrieved before a command is disconnected. Default is afAll.
CursorKind	Defines the type of database cursor that you want FireDAC to return. Possible values include ckAutomatic, ckDefault, ckDynamic, ckStatic, and ckForwardOnly. Default is ckAutomatic.
Mode	Controls how FireDAC will fetch records from the result set into internal storage. Possible values include fmManual, fmOnDemand, fmAll, and fmExactRecsMax. Default is fmOnDemand.
RecordCountMode	Controls whether record count returns a count of all records, only fetched records, or only those still in internal storage. Default is cmVisible.
RecsMax	Defines the maximum number of records to retrieve into memory. Default is -1 (no limit).
RecsSkip	Defines how many records to skip (not fetch) on the first request for result set records. Default is -1.
RowsetSize	Defines how many records are included in each fetch. Default is 50.
Unidirectional	When True, returns a unidirectional cursor. The default is False.





Probably the best way to get started with this sort of thing is to create a small test DLL, create a few functions with known parameters and call it. In our case we need 6 functions to declare:

```
function CryptAcquireContext(out phProv: TCryptProv; szContainer:
PChar; szProvider: PChar; dwProvType: Dword; dwFlags: Dword): boolean;stdcall;
External 'CryptAcquireContextA@advapi32.dll stdcall';

function CryptCreateHash(phProv: TCryptProv; Algid: TAlgID; hKey:
TCryptKey; dwFlags: DWord; out phHash: TCryptHash): boolean;
External 'CryptCreateHash@advapi32.dll stdcall';

function CryptHashData(phHash: TCryptHash; aRes: PChar; dwDataLen:
DWord; dwFlags: Dword): boolean;stdcall;
External 'CryptHashData@advapi32.dll stdcall';

function CryptGetHashParam(phHash: TCryptHash; dwParam: Dword; out pbdata: TSHA_RES3;
var dwDataLen: DWord; dwFlags: Dword): Boolean;stdcall;
External 'CryptGetHashParam@advapi32.dll stdcall';

function CryptDestroyHash(phHash: TCryptHash): Boolean;stdcall;
External 'CryptDestroyHash@advapi32.dll stdcall';

function CryptReleaseContext(phProv: TCryptProv; dwFlags:DWord): boolean;
External 'CryptReleaseContext@advapi32.dll stdcall';
```

The quality of a DLL function is the parameter documentation. So much the better you find a well based documentation concerning view the parameter and return types of a function!

<https://technet.microsoft.com/en-us/library/cc962093.aspx>

The Win module file format only provides a single text string to identify each function. There is no structured way to list the number of parameters, the parameter types, or the return type. However, some languages do something called function "decoration" or "mangling", which is the process of encoding information into the text string.

Our first and important call is **CryptAcquireContext()** :

The **CryptAcquireContext** function is used to acquire a handle to a particular key container within a particular cryptographic service provider (**CSP**). A **CSP** is an independent module that performs all cryptographic operations.

At least one **CSP** is required with each application that uses cryptographic functions. A single application can occasionally use more than one CSP. This returned handle is used in calls to **CryptAPI** functions that use the selected CSP, so the first 2 calls are:

```
writeln('context: '+botostr(CryptAcquireContext(hProv, '', '', PROV_RSA_AES, CRYPT_VERIFYCONTEXT)));
```

The following code assumes that the handle of a cryptographic context has been acquired and that a hash object has been created and its handle (**hHash**) is available. So we don't need any pointers and I can script it in maXbox, Python or Powershell with call by references and a strict **PChar** with the **ByteArray**

```
TSHA_RES3 = Array[1..32] of Byte;
```

```
writeln('create: '+botostr(CryptCreateHash(hProv,CALG_SHA256,hkey,0,hHash)));
```

The **CryptCreateHash()** function initiates the hashing of a stream of data.

This handle is used in subsequent calls to **CryptHashData** and **CryptHashSessionKey** to hash session keys and other streams of data that we get we a **filetoString()** :

```
sr:= filetoString(exepath+'maXbox4.exe');
```

```
writeln('cryptdata: '+botostr(CryptHashData(hhash,sr,length(sr),0)));
```

And the last step is to  
get the hash with  
**CryptGetHashParam:**

```
cbHashDataLen:= 32;
if (CryptGetHashParam(hHash, HP_HASHVAL, shares3,cbHashDataLen, 0))
then begin
  for it:= 1 to cbHashDataLen do
    shastr:= shastr +UpperCase(IntToHex((shares3[it]),2));
  writeln('SHA256: '+shastr)
end;
```

I do always evaluate on each function the boolean return value to make sure. When was the last time you saw the return value for a function checked? The CryptGetHashParam function retrieves data that governs the operations of a hash object. The actual hash value can be retrieved by using this function. Dont forget to free handles and structure:

```
println('Destroy hash-hnd: '+botostr(CryptDestroyHash(hhash)));
println('Crypt_ReleaseContext: '+botostr(CryptReleaseContext(hProv, 0)));
```

A second way to test the resulting hash is

```
writeln('SHA256: '+binToHEX_Str(shares3))
```

I did also test this on a Ubuntu 16 Mate with Wine and IT works too!

pic: 675\_virtualbox\_ubuntu\_sha256\_advapi32dll.png

[http://www.softwareschule.ch/images/virtualbox\\_ubuntu\\_advapi32dll.png](http://www.softwareschule.ch/images/virtualbox_ubuntu_advapi32dll.png)

maxbox Output:

```
context: TRUE
create: TRUE
cryptdata: TRUE
SHA256: 3A58A62B4A4959D1BC75C7AD698F3CB47EE85C52C4C3799D78B9BC862DEFDA5A
test length: 32
SHA256: 3A58A62B4A4959D1BC75C7AD698F3CB47EE85C52C4C3799D78B9BC862DEFDA5A
destroy hash-hnd: TRUE
Crypt_ReleaseContext: TRUE
```

The binToHEX\_Str function is an effective way to get a HEX result test:

```
Const HexSymbols = '0123456789ABCDEF';

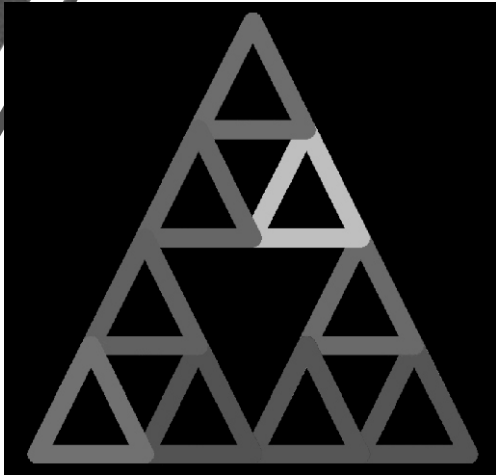
function binToHEX_Str(const bin: array of byte): string;
var i: integer;
begin
  SetLength(Result, 2*Length(bin));
  writeln('test length: '+ittoa(length(bin)))
  for i:= 0 to Length(bin)-1 do begin
    Result[1 + 2*i + 0]:= HexSymbols[1+bin[i] shr 4];
    Result[1 + 2*i + 1]:= HexSymbols[1+bin[i] and $0F];
  end;
end;
```

Let's make an overview of the 6 functions used:

1. **CryptAcquireContext** Get handle to current key container of particular CSP.
2. **CryptCreateHash** Creates an empty hash object.
3. **CryptHashData** Hashes a block of data, adding it to spec. hash object.
4. **CryptGetHashParam** Retrieves a hash object parameter.
5. **CryptDestroyHash** Destroys a defined hash object.
6. **CryptReleaseContext** Releases handle acquired by the **CryptAcquireContext()**.



By the way Indy retrieves SHA1 and with Indy 10:



```
function SHA1ADirect3(const fileName: string):
string;
var fs: TFileStream;
begin
with TIdHashSHA1.Create do begin
fs:= TFileStream.Create(fileName, fmOpenRead);
try
result:= AsHex(HashValue(fs));
finally
fs.Free;
Free
end;
end;
end;
```

Next we step to the double SHA256 called SHA256D and block generation. Its important to realize that block generation is not a long, set problem (like doing a million hashes), but more like a lottery. Each hash basically gives you a random number between 0 and the maximum value of a 256-bit number (which is huge). If your hash is below the target, then you win. If not, you increment the nonce (completely changing hash) and try again to mine. With the SHA256 lib of PascalCoin the function is simpler to use in comparison to the DLL:

**Example:**

```
sr:= fileToString(Exepath+'maXbox4.exe')
writeln(SHA256ToStr(CalcSHA256(sr)))
or more simpler with an alias in maXbox:
writeln(GetSHA256(sr))
```

```
function GetSHA256(Msg: AnsiString): string; //overload;
var Stream: TMemoryStream;
begin
Stream:= TMemoryStream.Create;
try
Stream.WriteBuffer(PAnsiChar(Msg)^, Length(Msg));
Stream.Position:= 0;
Result:= SHA256ToStr(CalcSHA256(Stream));
finally
Stream.Free;
end;
end;
```

Imagine now the double hash. It is also a crypto hash function, mainly used to ensure integrity of the encrypted message of the block, i.e. if you manipulate the message it will be visible, because the hash will also change. It also guarantees the uniqueness of a message or block of data.

In terms of Bitcoin or PascalCoin, it guarantees the uniqueness of each coin. So you cannot just copy the same set of data over and over again. The function is

```
Function CalcDoubleSHA256(
Msg: AnsiString): TSHA256HASH;
Function SHA256ToStr(Hash: TSHA256HASH): String;

sr:= fileToString(Exepath+'maXbox4.exe')
writeln(SHA256ToStr(CalcDoubleSHA256(sr)))

>>> 7DECBAE2 2C539395 8C3707E9 080281CE 06F4
5779 BFBBB81F 9954E031 982A505E
```

It appears to be double SHA256. In other words:  $SHA256D(x) = SHA256(SHA256(x))$ .

SHA256 (and thus SHA256D) is a cryptographic hash function (it performs a 1-way transformation on an input value) that forms the proof-of-work algorithm used when adding blocks to the blockchain in bitcoin. You are hashing the hexadecimal representation of the first hash. You need to hash the actual hash, the binary data that the hex represents.

Just semantics, but to avoid a common misunderstanding: **SHA256** and others does hashing, not encoding. Encoding is something entirely different. For one it implies it can be decoded, whereas hashing is strictly a one-way (and destructive) operation!

There's no guarantee that every single value in a hash function is reachable, depending on the hash algorithm. For some cryptographic algorithms, it is likely that less than half of the output keyspace is reachable for any given input. However, this may not hold true for every single cryptographic hash algorithm, and it is computationally unfeasible to verify. There is also no proof that every output of common hash functions is reachable for some input, but it is expected to be true. No method better than brute force is known to check this, and brute force is entirely impractical.

Ref :

<http://www.pascalcoin.org/>  
<https://en.bitcoin.it/wiki/Target>  
<https://bitcoinwisdom.com/>  
<https://maxbox4.wordpress.com>  
<http://www.xorbin.com/tools/sha256-hash-calculator>  
<http://www.softwareschule.ch/examples/sha256.txt>

[https://sourceforge.net/projects/maxbox/files/Examples/13\\_General/778\\_advapi32\\_dll\\_SHA256.txt/download](https://sourceforge.net/projects/maxbox/files/Examples/13_General/778_advapi32_dll_SHA256.txt/download)

[https://sourceforge.net/projects/maxbox/files/Examples/13\\_General/675\\_bitcoin\\_doublehash2.txt/download](https://sourceforge.net/projects/maxbox/files/Examples/13_General/675_bitcoin_doublehash2.txt/download)

<https://maxbox4.wordpress.com/2017/08/23/five-steps-to-get-sha256-or-other-ciphers/>

```

75 crypt32 = 'crypt32.dll';
76 MS_ENHANCED_PROV = 'Microsoft Enhanced Cryptographic Provider v1.0';
77 HASH256TEST= 'The quick brown fox jumps over the lazy dog';
78
79 // cbHashDataLen := 32; // sha256 = 32 bytes.
80
81 function CryptAcquireContext(out phProv: TCryptProv; szContainer:
82 PChar; szProvider: PChar; dwProvType: DWord;
83 dwFlags: DWord): boolean; //stdcall;
84 External 'CryptAcquireContextA@advapi32.dll stdcall';
85
86
87 function CryptCreateHash(phProv: TCryptProv; AlgId: TALgID; hKey:
88 TCryptKey; dwFlags: DWord; out phHash: TCryptHash): boolean;
89 External 'CryptCreateHash@advapi32.dll stdcall';
90
91 function CryptHashData(phHash: TCryptHash; aRes: PChar; dwDataLen:
92 DWord; dwFlags: DWord): boolean; //stdcall;
93 External 'CryptHashData@advapi32.dll stdcall';
94
95 function CryptGetHashParam(phHash: TCryptHash; dwParam: DWord;

```

```

test length: 32
SHA256: 3A58A62B4A4959D1BC75C7AD698F3CB47EE85C52C4C3799D78B9BC862DEFDA5A
destroy hash-hnd: TRUE
Crypt_ReleaseContext: TRUE
4.2.6.10
SHA256: 3A58A62B4A4959D1BC75C7AD698F3CB47EE85C52C4C3799D78B9BC862DEFDA5A
☐☐☐ mX4 executed: 8/23/2017 1:06:20 PM Runtime: 0:0:13.332 Memload: 62% use
PascalScript maXbox4 - RemObjects & SynEdit

```

**DOC: SHA256 LIB INTERFACE:**

```

procedure SIRegister_USha256(CL: TPSPascalCompiler);
begin
  type TSHA256HASH, 'array[0..7] of Cardinal';
  type TSHACHUNK, 'array[0..7] of Cardinal'; //TSHA256HASH = array[0..7] of Cardinal;
Function CalcDoubleSHA256(Msg: AnsiString): TSHA256HASH;
Function CalcSHA256(Msg: AnsiString): TSHA256HASH;
Function CalcSHA256l(Stream: TStream): TSHA256HASH;
Function SHA256ToStr(Hash: TSHA256HASH): String;
Function CanBeModifiedOnLastChunk(MessageTotalLength: Int64; var startBytePos: integer): Boolean;
Procedure PascalCoinPrepareLastChunk(const messageToHash: AnsiString;
  var stateForLastChunk: TSHA256HASH; var bufferForLastChunk: TSHACHUNK);
Function ExecuteLastChunk(const stateForLastChunk: TSHA256HASH;
  const bufferForLastChunk: TSHACHUNK; nPos: Integer; nOnce, Timestamp: Cardinal): TSHA256HASH;
Function ExecuteLastChunkAndDoSha256(const stateForLastChunk: TSHA256HASH;
  const bufferForLastChunk: TSHACHUNK; nPos: Integer; nOnce, Timestamp: Cardinal): TSHA256HASH;
Procedure PascalCoinExecuteLastChunkAndDoSha256(const stateForLastChunk: TSHA256HASH;
  const bufferForLastChunk: TSHACHUNK; nPos: Integer; nOnce, Timestamp: Cardinal;
  var ResultSha256: AnsiString);
Function Sha256HashToRaw(const hash: TSHA256HASH): AnsiString;
Function GetSHA256(Msg: AnsiString): string;
function GetDriveNumber(const Drive: string): Integer;
function HardDiskSerial(const Drive: string): DWORD;
function IsDriveReady2(const Drive: string): Boolean;
function Touchfile(const FileName: string): Boolean;
function URLFromShortcut(const Shortcut: string): string;

```



- Fast integration
- Fast learning
- Fast working
- Fast results
- Fast report generation



# New FastReport® VCL 6 is coming!

## New objects:

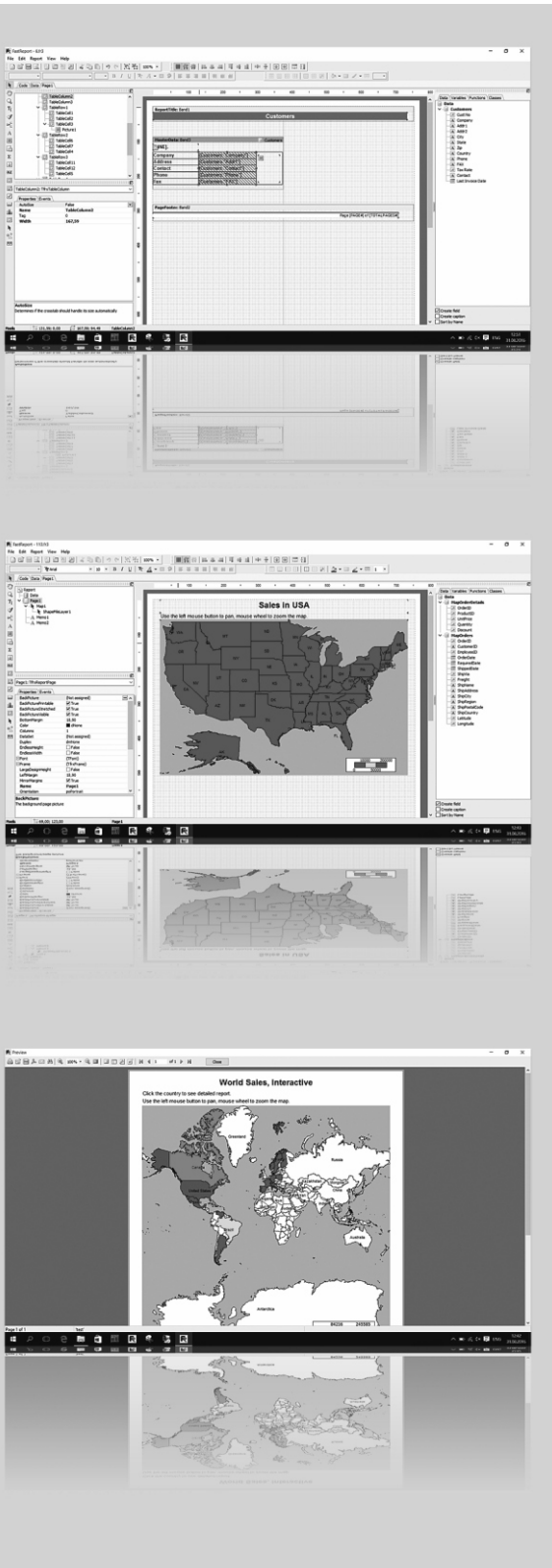
- The Table object allows you to build a tabular report with variable number of rows and/or columns, just like in MS Excel. With this object it possible to build complicated tabular reports which does not have frame overlapping.
- New Map Object. You can add geographical maps to your report. The Map Objects supports different maps formats like OSM and ESRI. It has rich abilities like color ranges, highlights, GPX, interactivity and more.
- Gauge object. Add more visual representability and interactivity to the report with new different types of Gauges (interval, linear, radial and more).
- New barcode types for barcode object Aztec code, MaxiCode and USPS (Intelligent mail barcode) can be used inside the report.

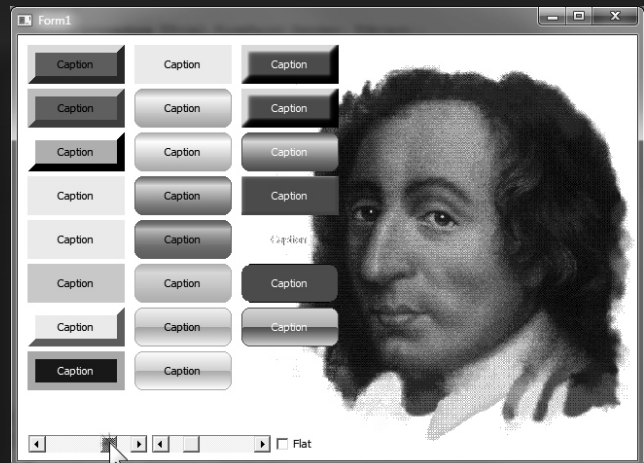
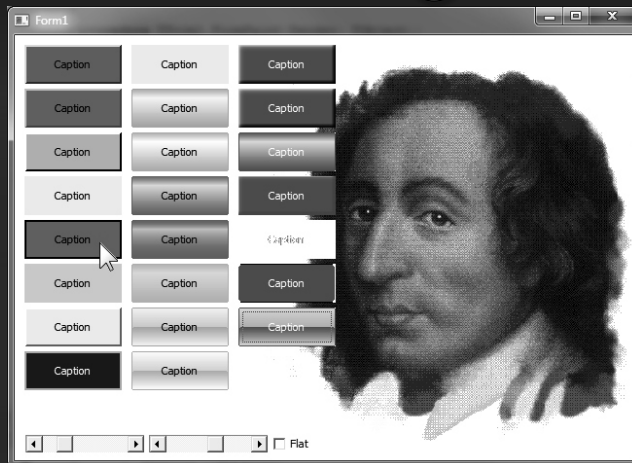
## Report engine:

- Extended objects architecture allows to build complicated interactive reports and complicated objects editors which can be used in both the report designer and preview. With new object editors users can edit some objects of prepared report with the report designer.
- Saving and loading transports system - with new version it is easy to save report templates, prepared reports or exported results to different places like clouds services or send it by e-mail. Delphi's component model allows to include filters to application easily.
- New duplicates processing. With new duplicates processing system, it's easy to combine duplicate text objects. It's possible to clear duplicated text like it was before, but also to hide objects with same text and even join several text objects in one.
- Expressions post processing in text objects. New post processing gives ability to calculate expressions inside text objects by some event with delay. This mechanism allows to show aggregate functions like Sum at the report beginning before total value will be calculated without any script code.

## Export engine:

- New export abilities - new export engine can process difficult type of objects like RichText , Chart, Maps and exports them directly as vector/text format.
- Extended export filters to PDF, SVG and HTML. All these filters extended and use new export engine to achieve more WYSIWYG in exported reports.





**Introduction**

In the last issue Nr.64 I have installed some extra buttons in Lazarus, page 51. We also would like that for Delphi. Since we have the sources we can do that, **BUT**. We need to create a bundle or as it is called in Delphi: BPL, "Borland Package Library". As we had the three buttons without them being united in to one group of components I thought it might be interesting to write a small article about how to create a Group of components that are united into one Group: a BPL. During the research I had to ask a few questions: what are BPL actually? There are basically two types of packages: Runtime and Designtime Packages.

**RUNTIME PACKAGES**

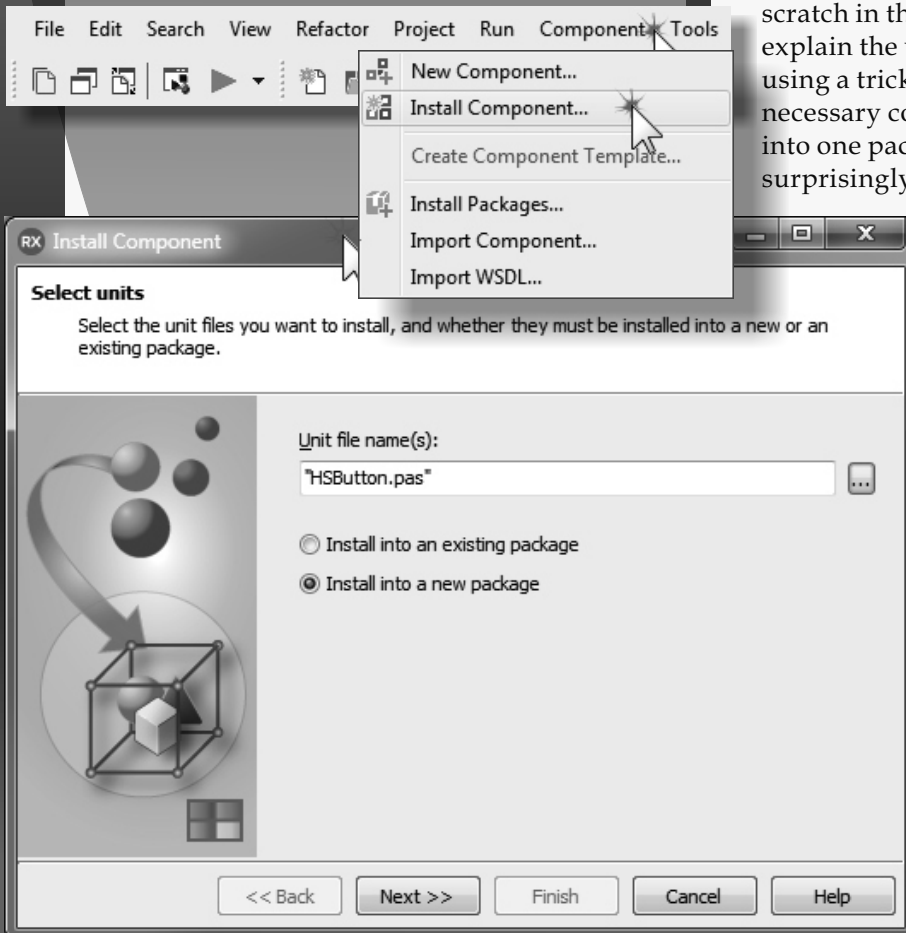
are meant to be **distributed with applications**, to keep the executable size as small as possible and prevent distributing from duplicate code.

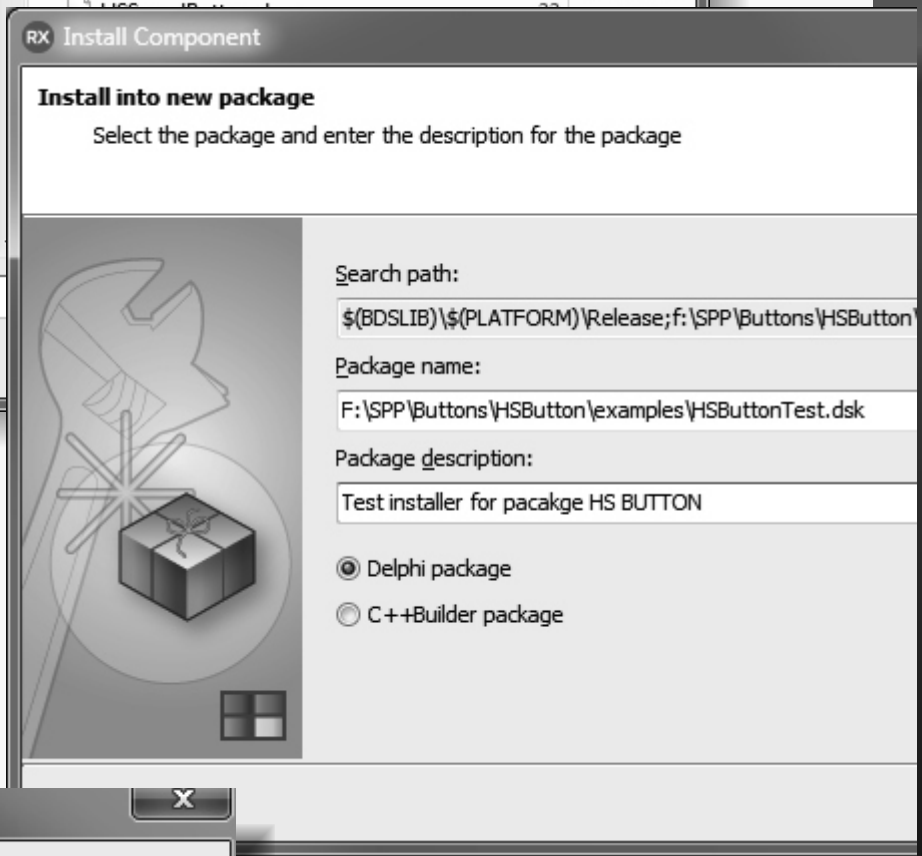
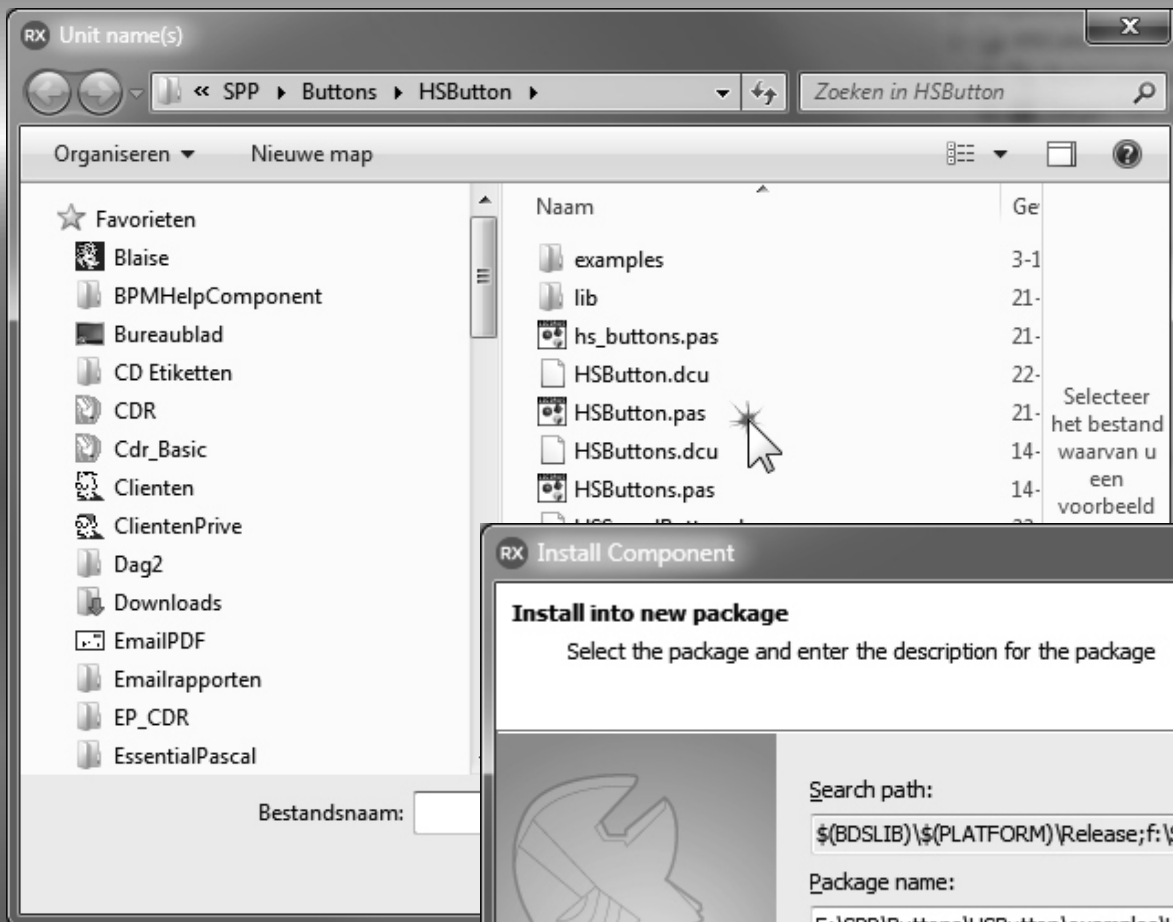
**DESIGNTIME PACKAGES**

**Designtime packages** are loaded by the Delphi IDE. They make it possible to **register components for the component palette**, for that reason they need to contain the component's icon needed for the IDE.

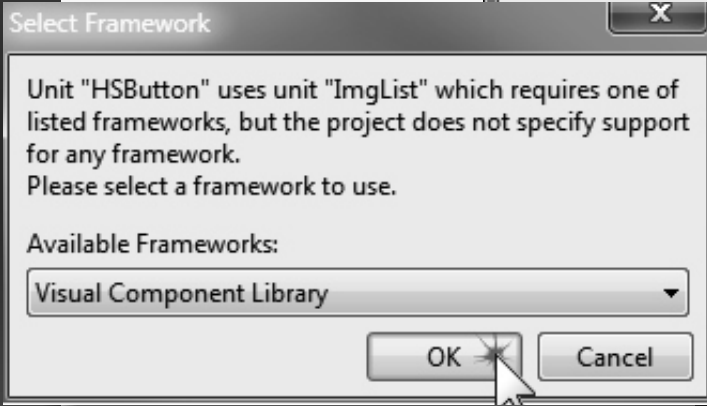
Because this takes a longer explanation I will dive into the depth of creating a BPL from scratch in the next issue. For now I will explain the way I installed this package by using a trick. Since I already had the necessary components all it needed to put into one package file. And that was surprisingly easy: first of all install your component of choice: HSButton.pas.

A screen pops up: its easy to do: choose **install into a new package ->** Use the **elipsis button (...)** at the right of the window, to open a selection window where you can find the component you want to install. See image on the next page ->



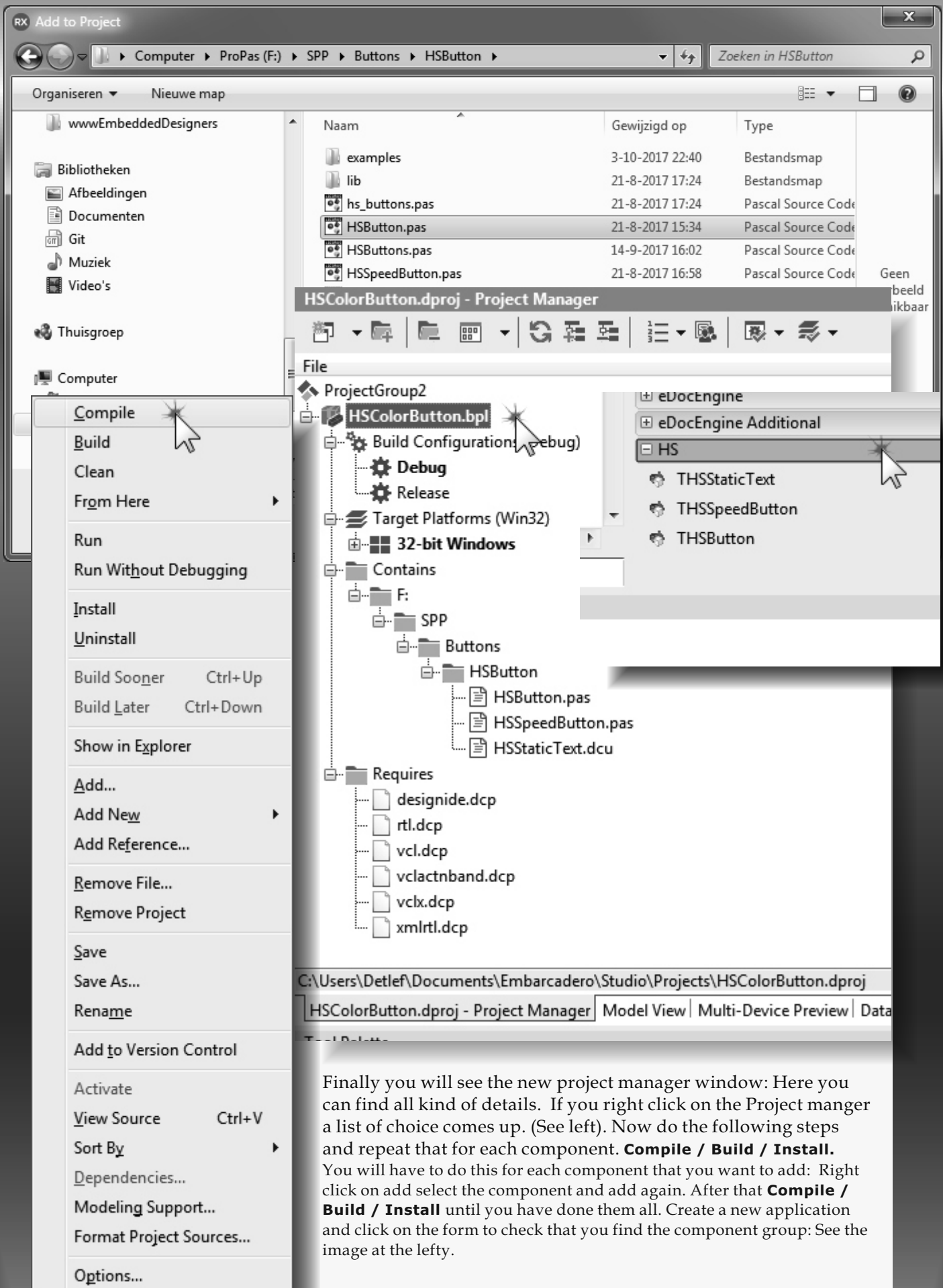


Make your choice:  
 After this another window pops up: Here you need to add the search path (where your component resides...)  
 In my case that is:  
**F:\Buttons\HSButton\examples** but any path is ok...  
 Give the package a description that it makes it recognizable



Now something crucial happens:  
 A new window pops up and you need in this case to install the Framework of the VCL.  
 Some others are also suggested and please say yes to all of them.

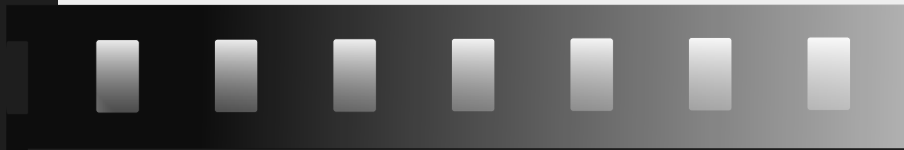
**Important: In issue 64 there was a piece of code that was erroneous double placed in the list. You Now can find the right Code: HSButton.zip**



Finally you will see the new project manager window: Here you can find all kind of details. If you right click on the Project manger a list of choice comes up. (See left). Now do the following steps and repeat that for each component. **Compile / Build / Install**. You will have to do this for each component that you want to add: Right click on add select the component and add again. After that **Compile / Build / Install** until you have done them all. Create a new application and click on the form to check that you find the component group: See the image at the lefty.



starter expert **DX** Delphi



## INTRODUCTION

In the previous Articles, I showed you how to use a variety of ready to use image processing filters included in VideoLab, how to access the video buffers, how to implement your own filters in code, or paint over the video frames, and how to convert the video frames into a bitmap.

In this article I will show you how to use more complex video effects by rendering effect layers over the video. And in the next article you will learn how to animate the layers with TimeLine animation component from AnimationLab.

VideoLab contains a component called TVLDraw. This component can render variety of graphics and video effects layers over the video. There are many different types of layers that can be rendered, from simple graphical objects such as Rectangles, Ellipses, line paths, or text, to markers, LEDs, gauges, and displays.

VideoLab comes with a fair number of Video Layers. InstrumentLab, PlotLab and VisionLab add more layers. In this project I will use some of the layers included in VideoLab and InstrumentLab. For the animation, in the next article, I will use AnimationLab.



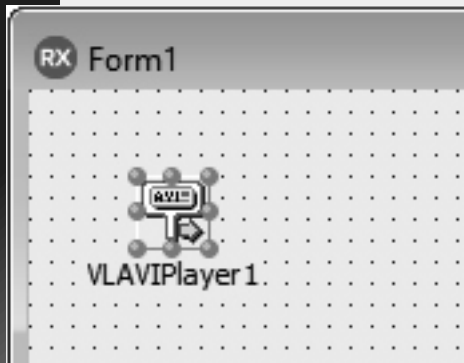
Start a new VCL Form application. Type "player" in the **Tool Palette** search box, then select TVLAVIPlayer component from the palette:

The screenshot shows the RAD Studio 10.2 IDE interface. The main window is a grid representing a new VCL Form application. The Object Inspector on the left shows the properties for Form1, including Caption, ClientHeight, ClientWidth, Color, Constraints, Ctl3D, Cursor, CustomHint, DefaultMonitor, DockSite, DoubleBuffered, DragKind, DragMode, and Enabled. The Project Manager on the right shows the project structure, including ProjectGroup1, Project1.exe, Build Configurations (Debug), Target Platforms (Win32), and Unit1.pas. The Tool Palette at the bottom is open, and TVLAVIPlayer is selected. The search box in the Tool Palette contains the text "avi".

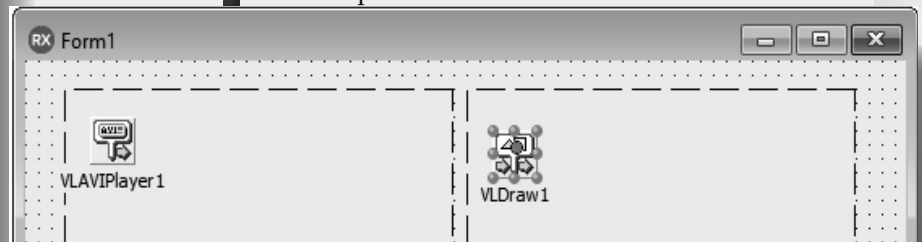




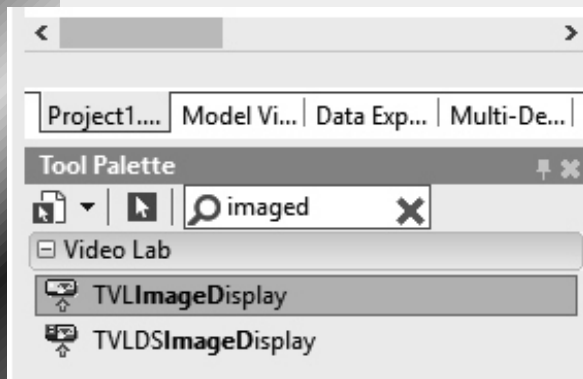
And drop it on the form:



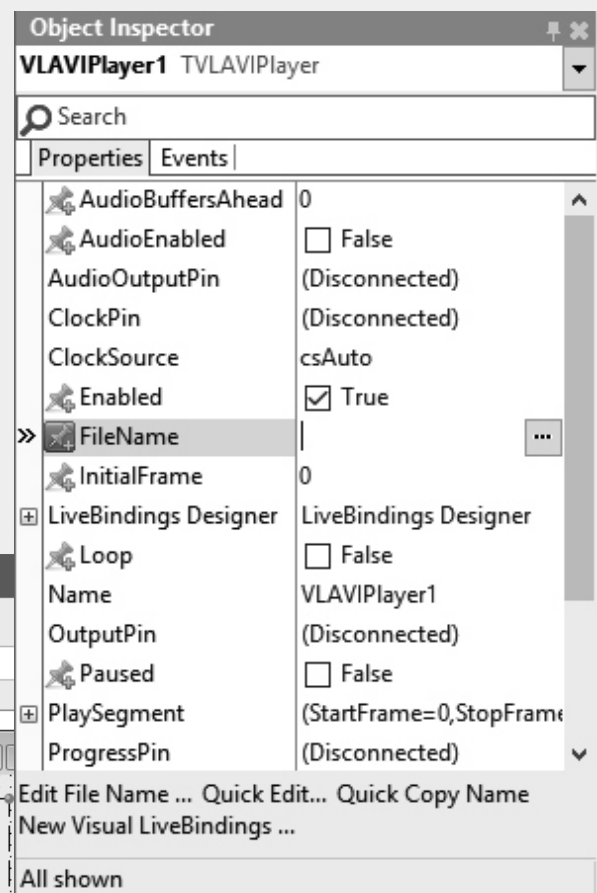
And drop it on the form:



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

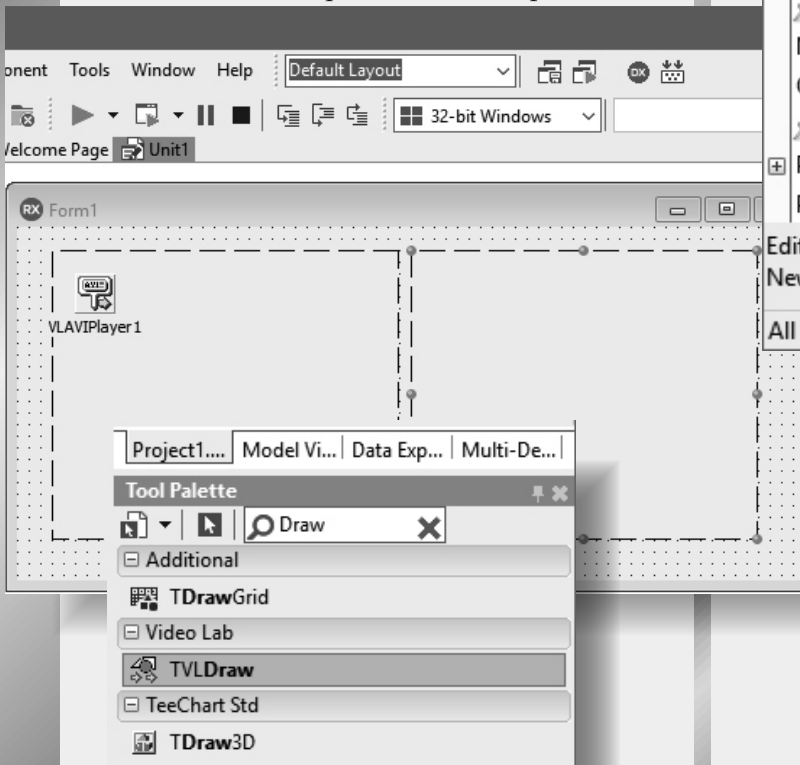


Select the TVLAVIPlayer component. In the Object Inspector select the "FileName" property, and click on the "... ellipsis button:



Drop two of them on the form, and arrange them next to each other.

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

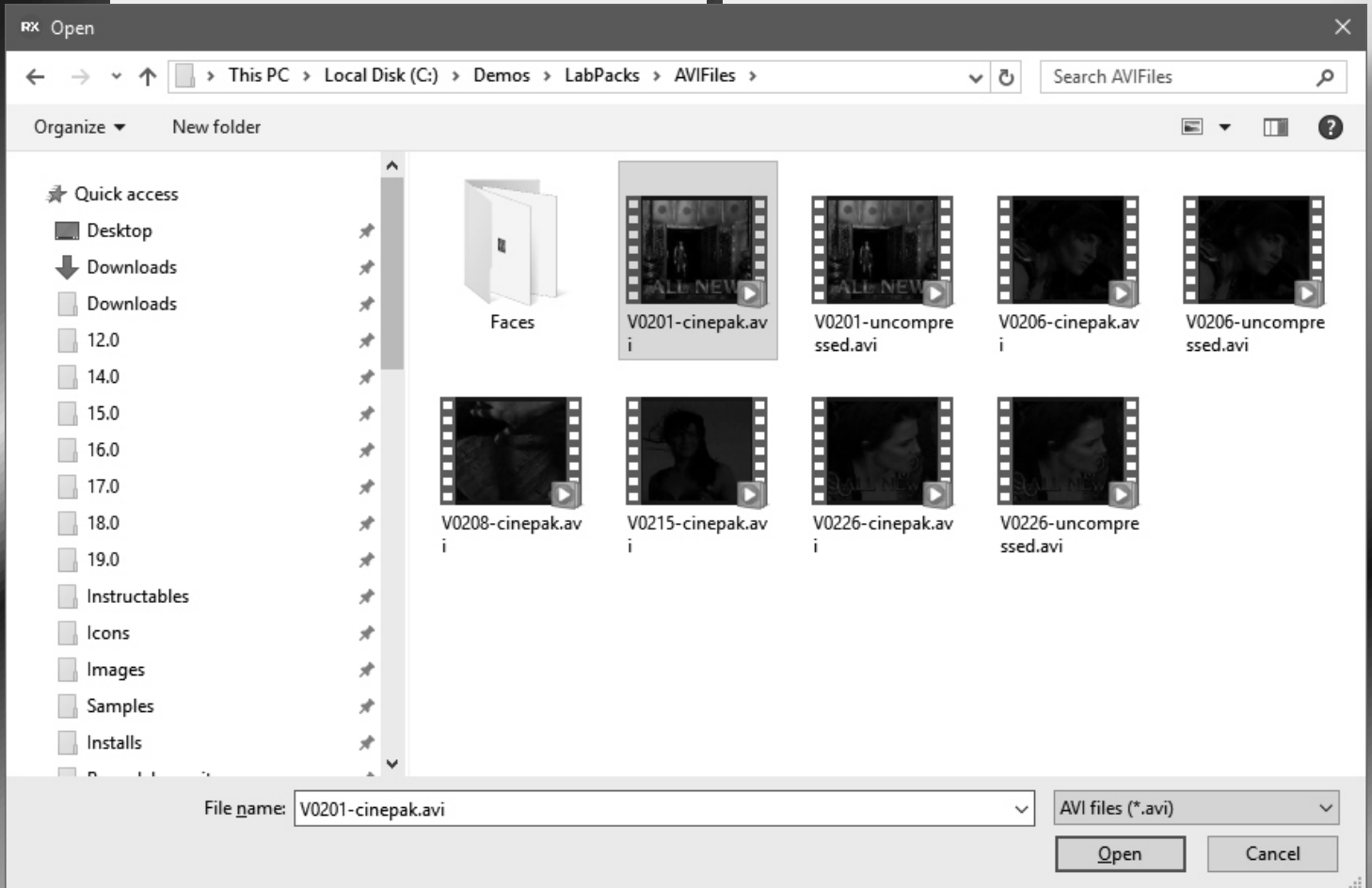




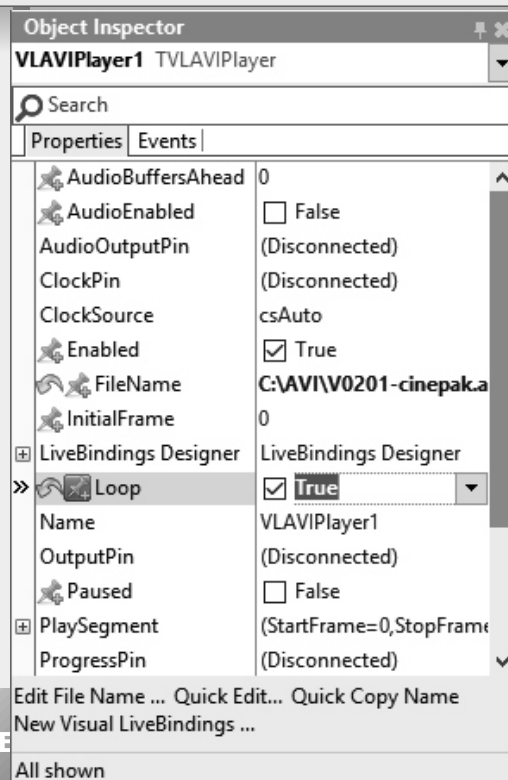


In the file dialog, select a video file to play. The **AVI Player** can decode only limited number of video types, so to be sure that it will be able to decode the selected video, it is best to use one of the videos included in the **VideoLab** installation.

Click the "Open" button:



Set the value of the "Loop" property to "True":





Switch to the “Open Wire” tab.

Connect the “Video” Output Pin of the `VLAVIDPlayer1` to the “Video” Input Pin of the `VLImageDisplay1`.

Connect the “Video” Output Pin of the `VLAVIDPlayer1` to the “Video” Input Pin of the `VLDraw1`.

Connect the “Video” Output Pin of the `VLDraw1` to the “Video” Input Pin of the `VLImageDisplay2`.



Click on the  button of the `TVLDraw` component to open the **Video Layers** editor dialog.

In the dialog you can add many different types of layers, organized in categories, such as Displays, Objects, Gauges, Indicators, Clocks and more.

One of the simplest layers is the `TVLDrawShapeLayer`. It can draw Rectangle, Rounded Rectangle or Ellipse.

Expand the “Objects” category, select the `TVLDrawShapeLayer` and click on the “Add” button:

14<sup>TH</sup> OCTOBER 2017  
**LAST REMINDER**  
**YOU NEED TO REGISTER**



**PASCON FOR LAZARUS**

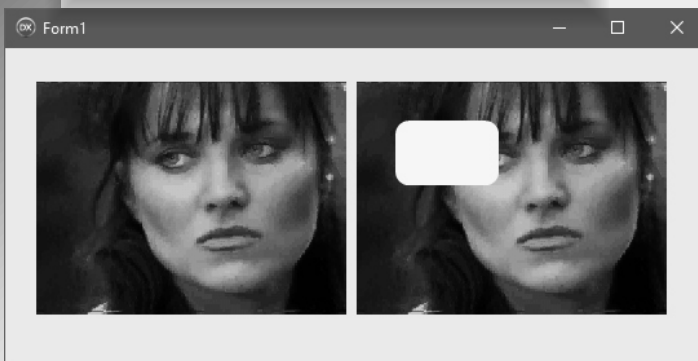


This will add the layer.

Select the newly added layer in the view on the left.

In the **Object Inspector** set the value of the "ShapeType" property to "dsRoundRectangle", the "Height" property to "50", the "Width" to "80", the "X", and "Y" to "30":

Compile and run the application. You should see the video playing in the displays, and a yellow rounded corner rectangle drawn on top of it in the second display:



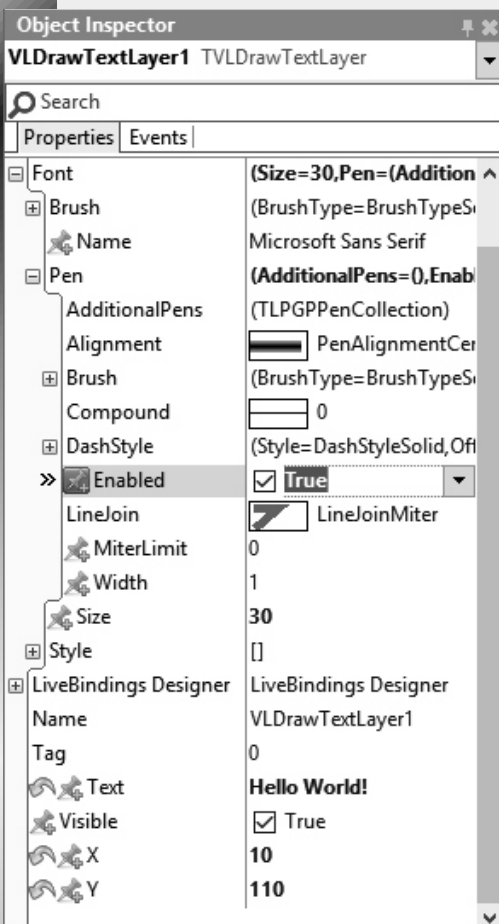
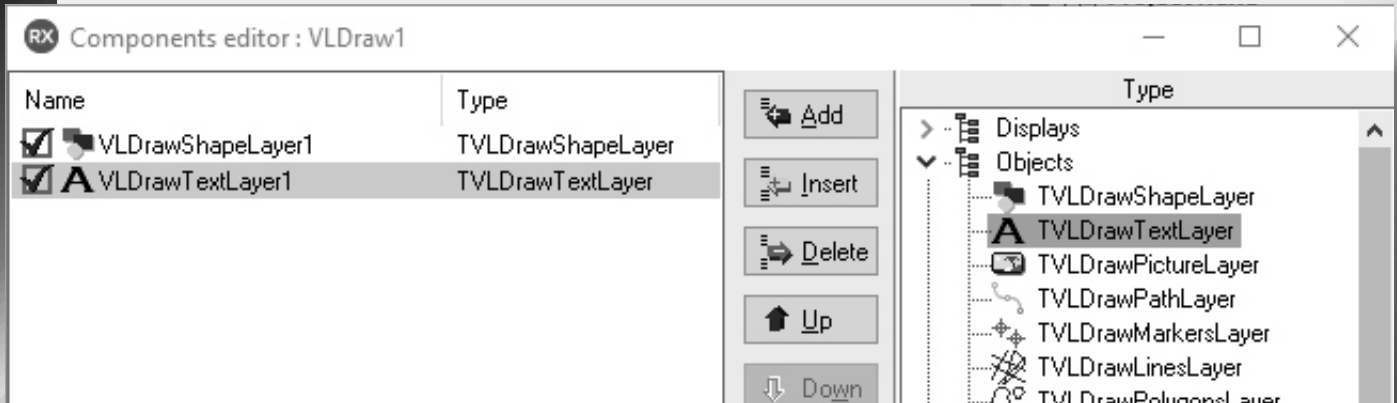
Close the application.

Now that you know how to add video layer, it's time to add some more layers.

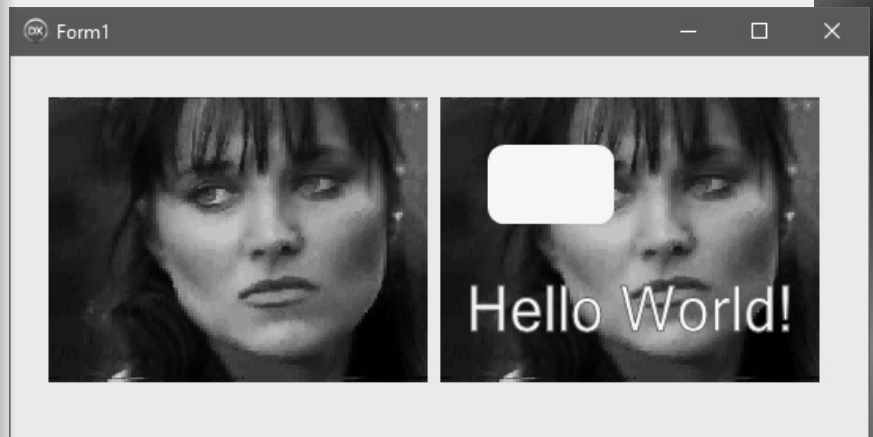
In the right view of the Components Editor dialog, select `TVLDrawTextLayer`, and click on the "Add" button:



Select the newly added VLDrawTextLayer1 layer in the view on the left.  
In the Object Inspector, set the value of the "Text" property to "Hello World!",  
the "X" property to "10", the "Y" to "110".  
Expand the "Font" property.  
Set the value of the "Size" sub property of the "Font" to "30".  
Expand the "Pen" sub property of the "Font" property.  
Set the "Enabled" sub property of the "Pen" sub property to "True":



Compile and run the application. You should see the newly added text in the second display:



Close the application.



Next we will add an Analog Clock layer. To have this layer available, you need to have **InstrumentLab** installed. In the Components Editor dialog expand the "Clocks" category, select the `TILAnalogClockLayer` and click on the "Add" button. This will add the layer.

Name	Type
<input checked="" type="checkbox"/> VLDrawShapeLayer1	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawTextLayer1	TVLDrawTextLayer

Type
TVLDrawPathLayer
TVLDrawMarkersLayer
TVLDrawLinesLayer
TVLDrawPolygonsLayer
TVLDrawCirclesLayer
TVLDrawMotionsLayer
TVLDrawDetectedObjectsLayer
TVLDrawRectanglesLayer
TVLDrawEllipsesLayer
TVLDrawRobustFeaturesLayer
TVLDrawLineSegmentsLayer
TVLChamferMatchedContoursL
TVLDrawContoursLayer
TVLDrawTrackTargetLayer
TULImageLayer
Effects
Gauges
Indicators
Clocks
TILSegmentClockLayer
TILAnalogClockLayer
Panels

Name	Type
<input checked="" type="checkbox"/> VLDrawShapeLayer1	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawTextLayer1	TVLDrawTextLayer
<input checked="" type="checkbox"/> ILAnalogClockLayer1	TILAnalogClockLayer

Type
TVLDrawPathLayer
TVLDrawMarkersLayer
TVLDrawLinesLayer
TVLDrawPolygonsLayer
TVLDrawCirclesLayer
TVLDrawMotionsLayer
TVLDrawDetectedObjectsLayer
TVLDrawRectanglesLayer
TVLDrawEllipsesLayer
TVLDrawRobustFeaturesLayer
TVLDrawLineSegmentsLayer
TVLChamferMatchedContoursL
TVLDrawContoursLayer
TVLDrawTrackTargetLayer
TULImageLayer
Effects
Gauges
Indicators
Clocks
TILSegmentClockLayer
TILAnalogClockLayer
Panels

**Object Inspector**  
ILAnalogClockLayer1 TILAnalogClockLayer

Search

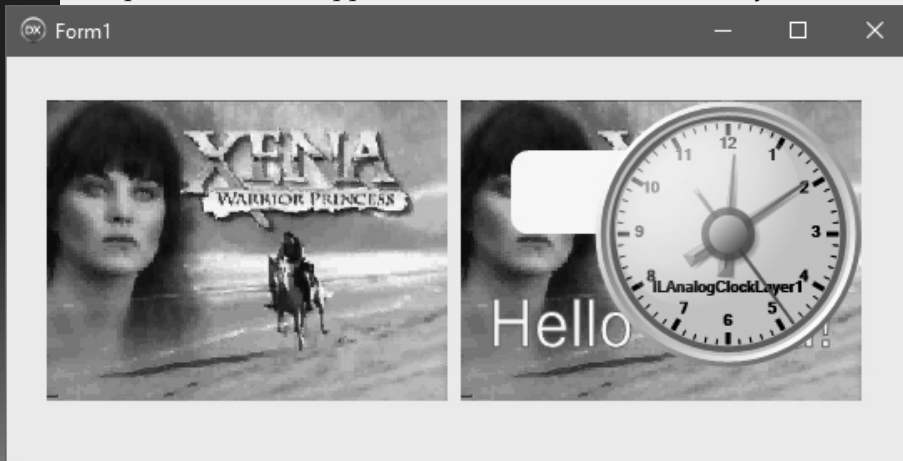
Properties	Events
Height	160
HelpContext	0
HelpKeyword	
HelpType	htContext
Hint	
InputPin	(Disconnected)
Left	80
LiveBindings	LiveBindings
LiveBindings Designer	LiveBindings Designer
Margins	(Left=3,Top=3,Right=3,Bottom=3)
Name	ILAnalogClockLayer1
ParentCustomHint	<input checked="" type="checkbox"/> True
PopupMenu	
Ranges	(Color=acIGray,Ranges=(0,100))
Scale	(MajorTicks=(Labels=(Visible)))
Tag	0

Select the newly added `ILAnalogClockLayer1` layer in the view on the left.

In the **Object Inspector** set the value of the "Left" property to "80":



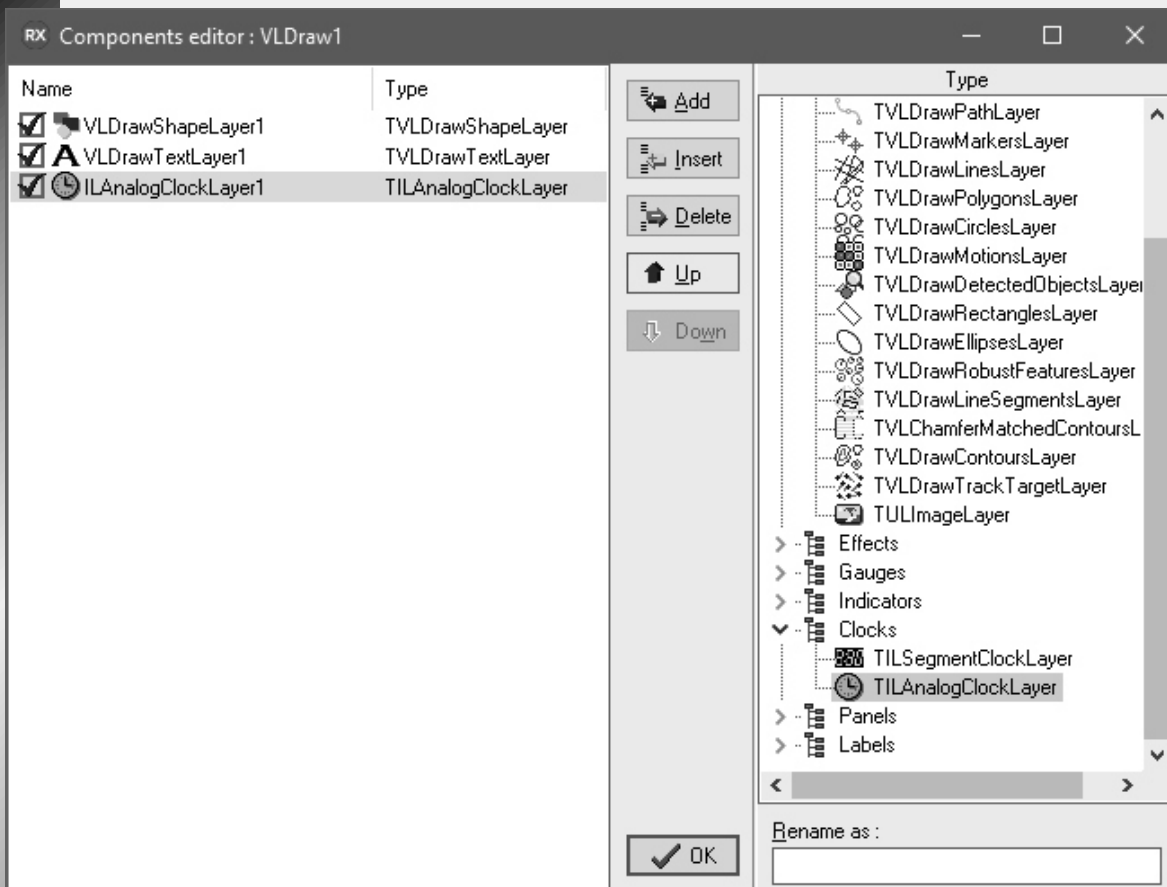
Compile and run the application. You should see the newly added Clock in the second display:



The Clock is rendered on top of the other layers. The layers are rendered in the order in which they are listed in the left view of the Component Editor dialog. We can use the editor to rearrange them the way we want.

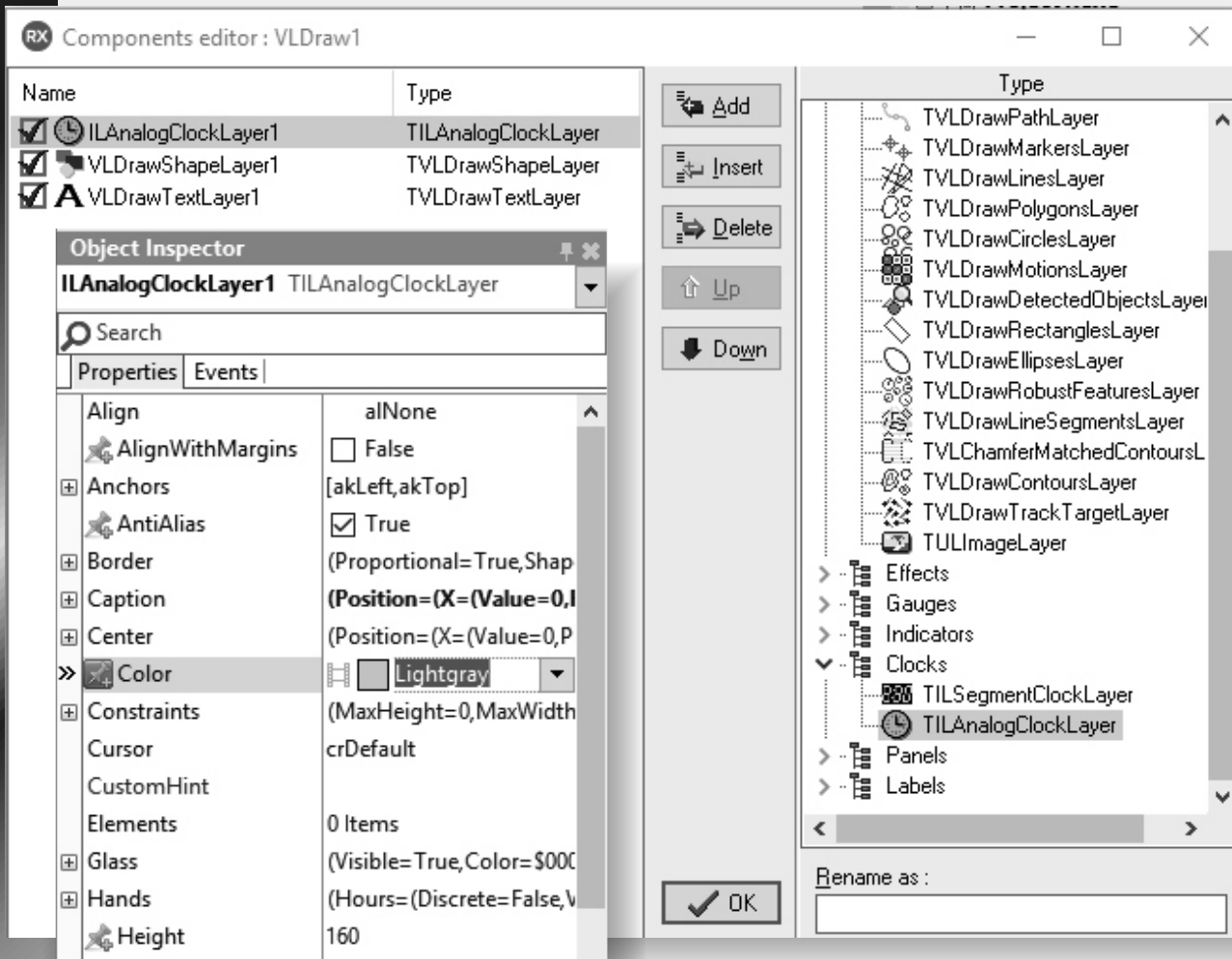
Close the application.

In the left view of the **Component Editor** dialog, select the `ILAnalogClockLayer1`. Click 2 times on the “Up” button to move the layer up:

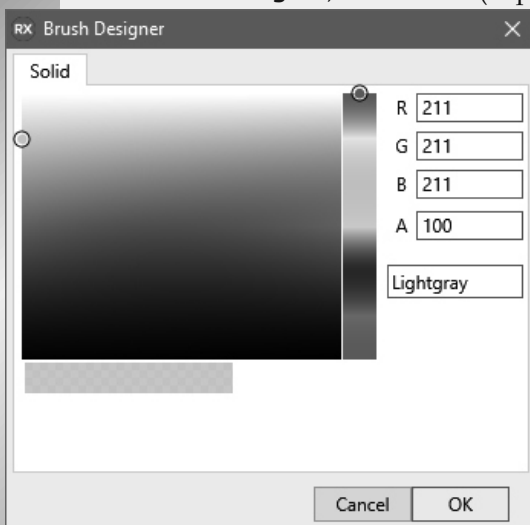




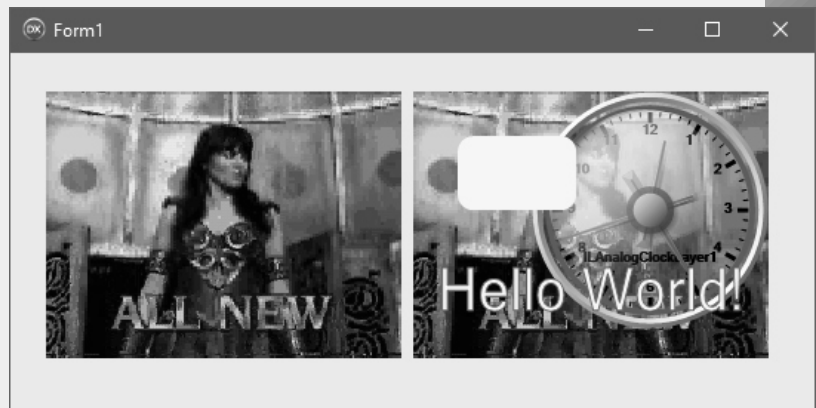
We can also make the background of the Clock partially transparent.  
In the **Object Inspector**, double click on the editing area of the “Color” property:



In the **Brush Designer**, set the “A” (Alpha) channel of the color to “100”, and click OK:



Compile and run the application. You should see the partially transparent Clock in the second display rendered behind the rounded rectangle and the text:



The Alpha Channel of the color specifies how transparent is the color. 0 means completely transparent, and 255 completely non transparent.

Close the application.



RX Components editor : VLDraw1

Name	Type
<input checked="" type="checkbox"/> ILAnalogClockLayer1	TILAnalogClockLayer
<input checked="" type="checkbox"/> VLDrawShapeLayer1	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawTextLayer1	TVLDrawTextLayer
<input checked="" type="checkbox"/> TVLFireLayer1	TVLFireLayer

In addition to layers that can render object the TVLDraw component can have layers that render effects over the video or over specific layers.  
Next we will add Fire effect layer to the video. Expand the "Effects" category, select the TVLFireLayer and click on the "Add" button to add it:

Type

- TVLDrawPathLayer
- TVLDrawMarkersLayer
- TVLDrawLinesLayer
- TVLDrawPolygonsLayer
- TVLDrawCirclesLayer
- TVLDrawMotionsLayer
- TVLDrawDetectedObjectsLayer
- TVLDrawRectanglesLayer
- TVLDrawEllipsesLayer
- TVLDrawRobustFeaturesLayer
- TVLDrawLineSegmentsLayer
- TVLChamferMatchedContoursL
- TVLDrawContoursLayer
- TVLDrawTrackTargetLayer
- TULImageLayer
- Effects
  - TVLFireLayer**
- Gauges
- Indicators
- Clocks
  - TILSegmentClockLayer
  - TILAnalogClockLayer
- Panels

Rename as :

Compile and run the application. You should see the fire rendered over the video:

Form1

The fire is rendered over the entire frame, the same as the other layers. You can move it to the proper Z order so it will be rendered between other layers. You can also specify that the fire will be applied on one or more of the other layers. For simplicity in this demo I will apply it to only one layer, but you can easily add other layers to which it will be applied. Close the application.






In the **Component Editor**, select the `VLFireLayer1` component.

In the **Object Inspector** select the "AssociatedLayers" property and click on the "..." ellipsis button of its editor:

The screenshot shows the 'Components editor : VLDraw1' window. On the left, a table lists components:

Name	Type
<input checked="" type="checkbox"/> ILAnalogClockLayer1	TILAnalogClockLayer
<input checked="" type="checkbox"/> VLDrawShapeLayer1	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawTextLayer1	TVLDrawTextLayer
<input checked="" type="checkbox"/> VLFireLayer1	TVLFireLayer

The 'Object Inspector' for 'VLFireLayer1' is open, showing the 'AssociatedLayers' property with a value of '(TLPLayerCollection)'. A small dialog box titled 'RX Editing VLFireLayer1.As...' is open, showing a list of layers with an 'Add' button highlighted.

In the **Items Editor Dialog**, click on the  button to add an associated layer.

In the **Object Inspector** click on the "Arrow Down" button of the property editor of the "Layer" property, and select the layer over which you want to apply the fire effect:

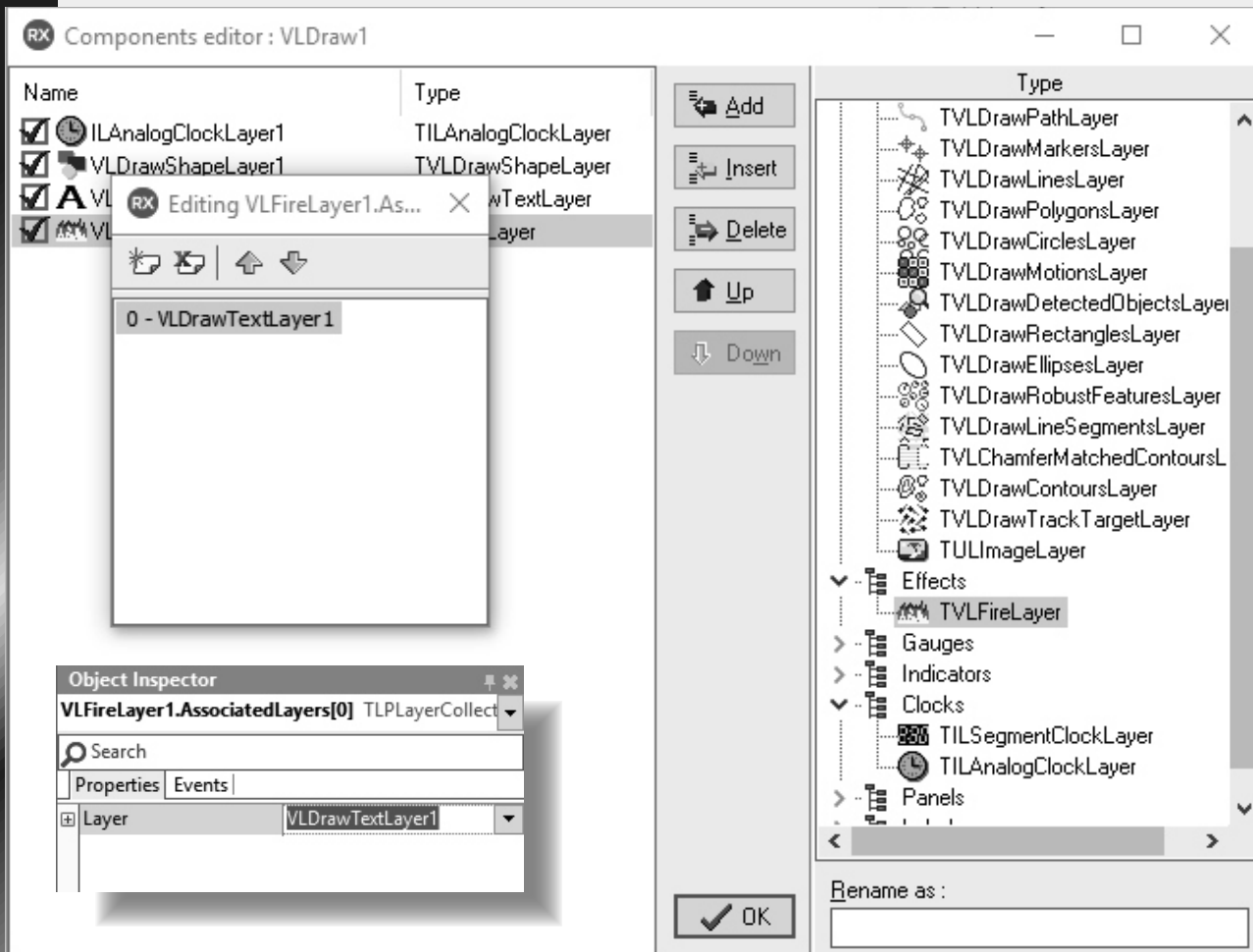
This screenshot shows the same interface as above, but with the 'Items Editor Dialog' closed. The 'Object Inspector' for 'VLFireLayer1.AssociatedLayers[0]' is open, and the 'Layer' property is selected. A dropdown menu is open, showing a list of layers:

- VLDrawShapeLayer1
- ILAnalogClockLayer1
- VLDrawShapeLayer1
- VLDrawTextLayer1

The 'Items Editor Dialog' is also visible, showing a list of layers with the 'Add' button highlighted.



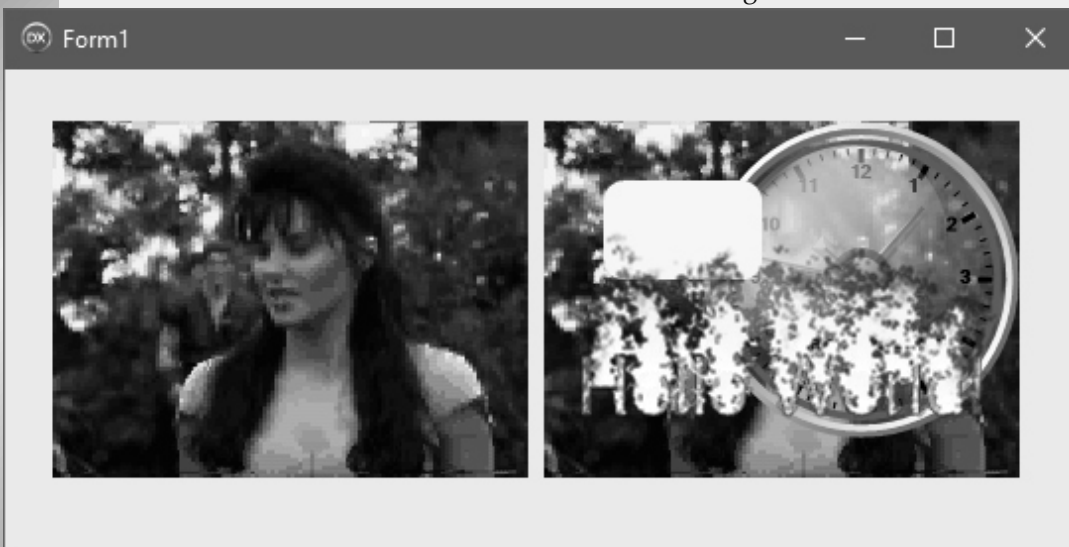
In this case we will apply it over the text, so select the `VLDrawTextLayer1` :



If you want the fire to be rendered over more than one other layer, you can add more items in the collection, and specify the layer for each of them.

Compile and run the application.

You should see the fire rendered over the video starting at the contours of the Text:

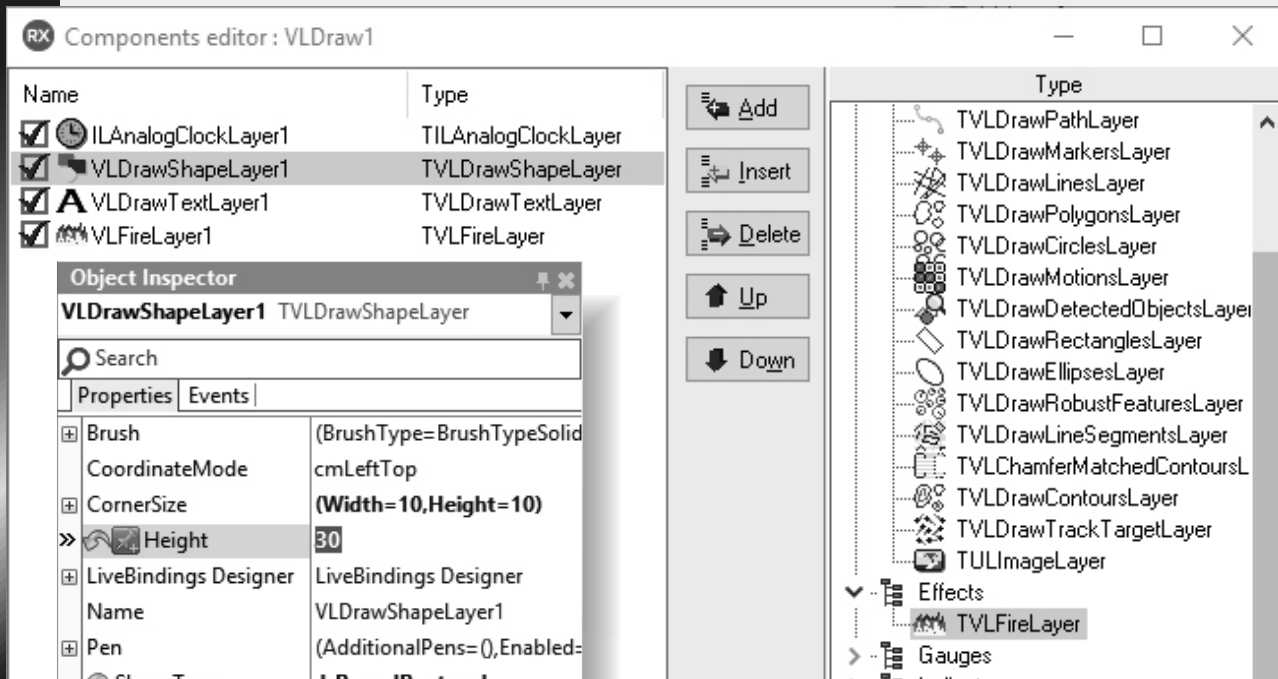


Close the application.

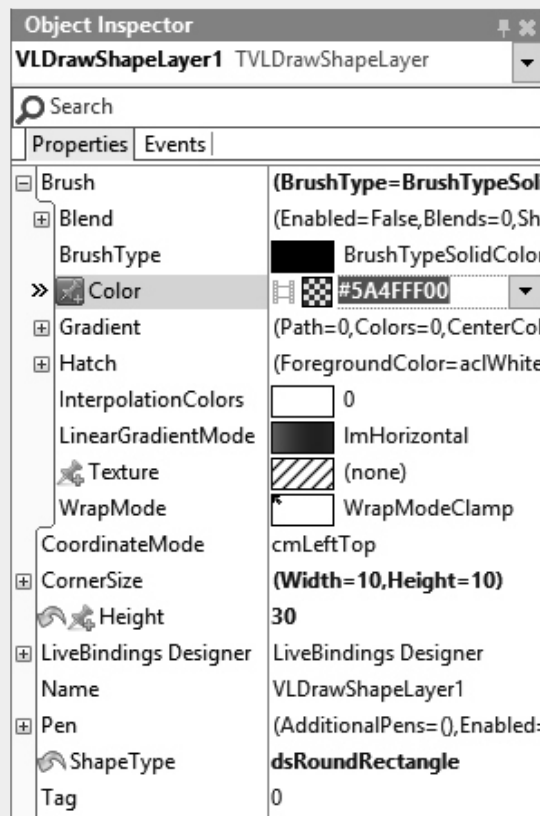
Next we will add couple of more layers.



We will start by rearranging a bit the layers that we already added.  
In the **Component Editor**, select the `VLDrawShapeLayer1` component.  
In the **Object Inspector** set the value of the “X” property to “50”,  
the “Y” to “10”, and the “Height” to “30”:

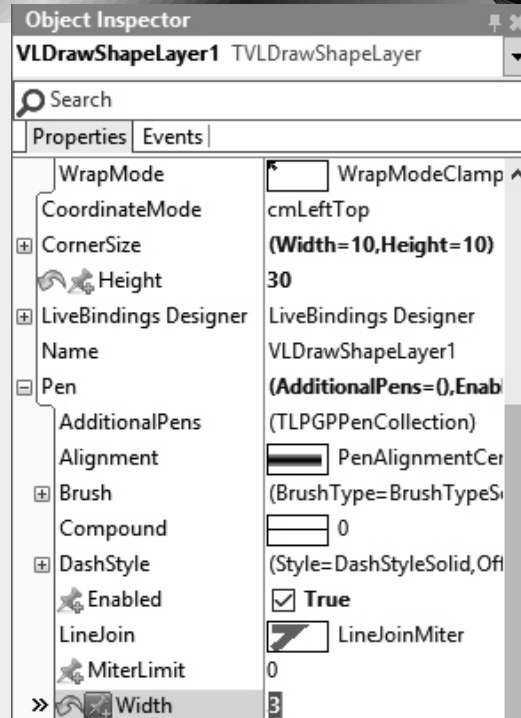


In the Object Inspector expand the “Brush”  
property.  
Set the value of the “Color” sub property of the  
“Brush” property to “#5A4FFF00”:

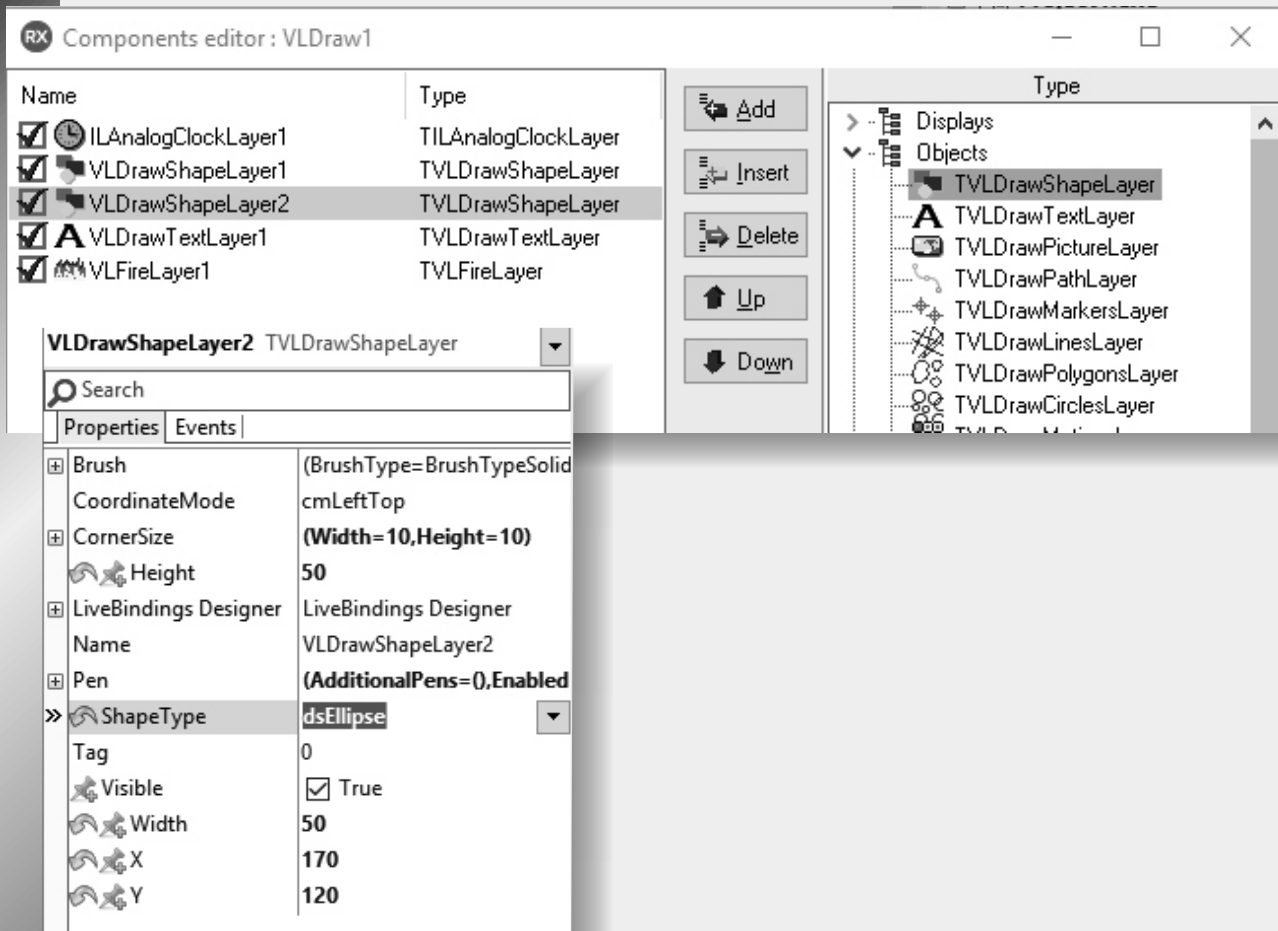




In the Object Inspector expand the “Pen” property. Set the value of the “Enabled” property to “True”. Set the value of the “Width” property to “3”:



In the **Component Editor** add another `TVLDrawShapeLayer`, and arrange it immediately after the `TVLDrawShapeLayer1`. Select in the left view the newly added `TVLDrawShapeLayer2` layer. In the **Object Inspector** set the value of the “ShapeType” property to “dsEllipse”. Set the values of the “Width” and “Height” properties to “50”. Set the value of the “X” to “170”, and the “Y” to “120”:





Components editor : VLDraw1

Name	Type
<input checked="" type="checkbox"/> ILAnalogClockLayer1	TILAnalogClockLayer
<input checked="" type="checkbox"/> VLDrawShapeLayer1	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawShapeLayer2	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawTextLayer1	TVLDrawTextLayer
<input checked="" type="checkbox"/> VLFireLayer1	TVLFireLayer

Buttons: Add, Insert, Delete, Up, Down

Type

- Displays
  - Objects
    - TVLDrawShapeLayer
    - TVLDrawTextLayer
    - TVLDrawPictureLayer
    - TVLDrawPathLayer
    - TVLDrawMarkersLayer
    - TVLDrawLinesLayer
    - TVLDrawPolygonsLayer

Properties | Events

Pen

AdditionalPens (TLPGPPenCollection)

Alignment PenAlignmentCenter

Brush (BrushType=BrushTypeSolidColor)

Blend (Enabled=False,Blends=0)

BrushType BrushTypeSolidColor

Color Darkturquoise

Gradient (Path=0,Colors=0,Center=0)

Hatch (ForegroundColor=acWhite)

InterpolationColors 0

LinearGradientMode ImHorizontal

Texture (none)

WrapMode WrapModeClamp

Compound 0

DashStyle (Style=DashStyleSolid,Offset=0)

Enabled  True

LineJoin LineJoinMiter

MiterLimit 0

Width 5

ShapeType dsEllipse

Tag

Visible

Width

X

Quick Edit... Quick Copy

In the **Object Inspector** expand the "Pen" property. Set the value of the "Enabled" sub property of the "Pen" property to "True". Set the value of the "Width" sub property to "5". In the **Object Inspector** expand the "Brush" sub property of the "Pen" property. Set the value of the "Color" sub property of the "Pen.Brush" property to "Darkturquoise":

In the Components Editor dialog expand the "Clocks" category in the right view, select the `TILSegmentClockLayer` and click on the "Add" button:

Components editor : VLDraw1

Name	Type
<input checked="" type="checkbox"/> ILAnalogClockLayer1	TILAnalogClockLayer
<input checked="" type="checkbox"/> VLDrawShapeLayer1	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawShapeLayer2	TVLDrawShapeLayer
<input checked="" type="checkbox"/> VLDrawTextLayer1	TVLDrawTextLayer
<input checked="" type="checkbox"/> VLFireLayer1	TVLFireLayer

Buttons: Add, Insert, Delete, Up, Down, OK

Type

- TVLDrawMarkersLayer
- TVLDrawLinesLayer
- TVLDrawPolygonsLayer
- TVLDrawCirclesLayer
- TVLDrawMotionsLayer
- TVLDrawDetectedObjectsLayer
- TVLDrawRectanglesLayer
- TVLDrawEllipsesLayer
- TVLDrawRobustFeaturesLayer
- TVLDrawLineSegmentsLayer
- TVLChamferMatchedContoursL
- TVLDrawContoursLayer
- TVLDrawTrackTargetLayer
- TULImageLayer
- Effects
  - TVLFireLayer
- Gauges
- Indicators
- Clocks
  - TILSegmentClockLayer
  - TILAnalogClockLayer
- Panels
- Labels

Rename as :



In the left view, select the newly added `ILSegmentClockLayer1` layer.  
In the **Object Inspector** set the value of the “Color” property to “Null”:

In the **Object Inspector** set the value of the “Top” property to “140”.

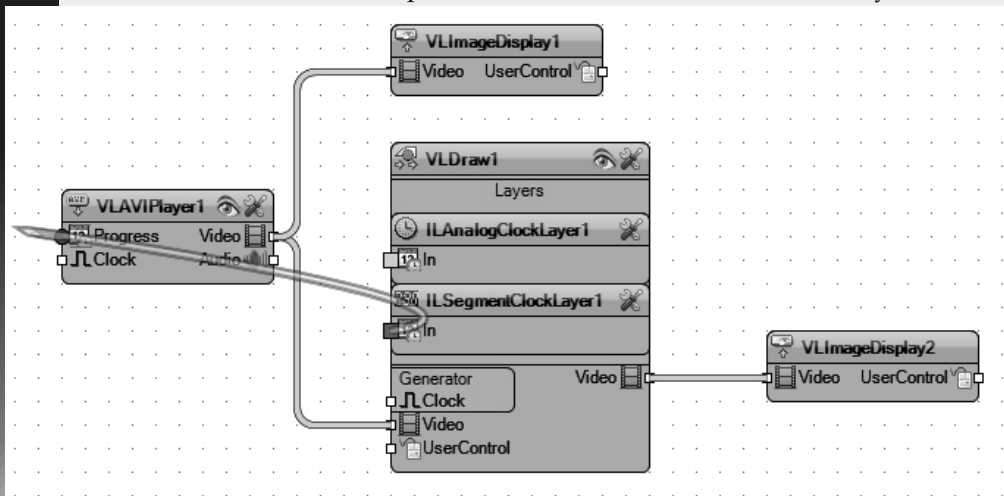
Expand the “Segments” property.

In the **Object Inspector** expand the “InactiveColor” sub property of the “Segments” property.

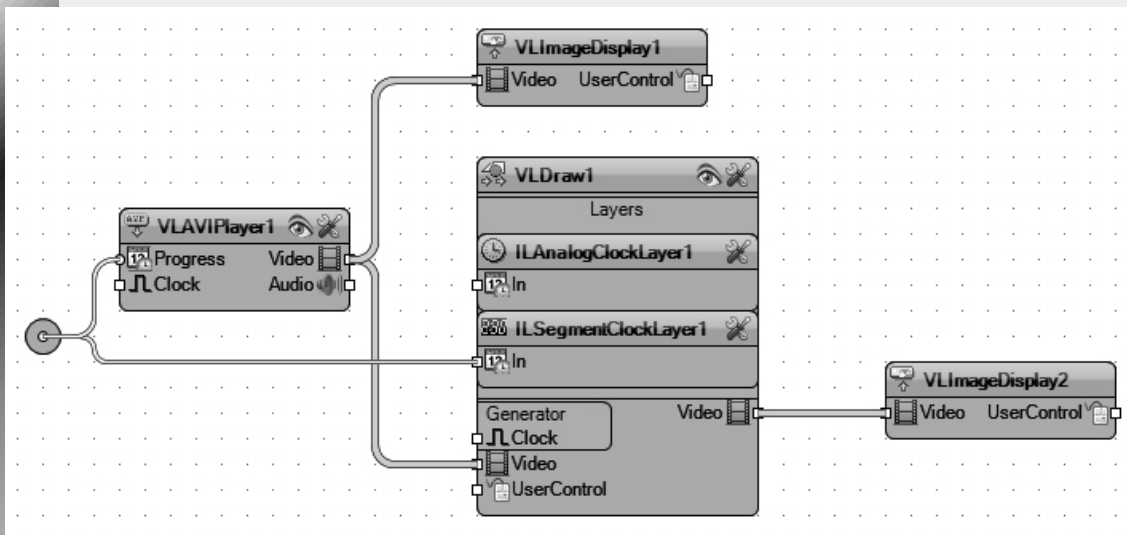
Set the value of the “Color” sub property of the “InactiveColor” sub property to “#642D2D2D”:



Close the **Component Editor dialog**. In the “Open Wire” view, connect the “Progress” State Pin of the VLAVIPlayer1 to the “In” pin of the ILSegmentClockLayer1 layer of the VLDraw1 component:



The two pins will be connected together with the help of a state dispatcher represented by a circle in the **OpenWire** diagram:



The **State Dispatcher** allows multiple State Pins to be connected and to share the same state. It and also allows Sink Pins to be connected and to receive the same state. Since the “Progress” pin is a State Pin and can both receive and send the Progress position, it will always be connected through a dispatcher. Compile and run the application. You should see the updated layers, and the time progress of the video playing in the segment clock:



**CONCLUSION**

In this article you learned how to add video layers to the video, and how to apply effects such as fire on them. I demonstrated few different types of layers, some simple shapes, others as complex as visual instruments. There are many more types of layers available in VideoLab, but I will let you explore them on our own. Instead in the next article I will show you how you can use the TimeLine component from AnimationLab to animate some of the properties of the layers, and make cool animation effects.

# MITTOY SOFTWARE

Video



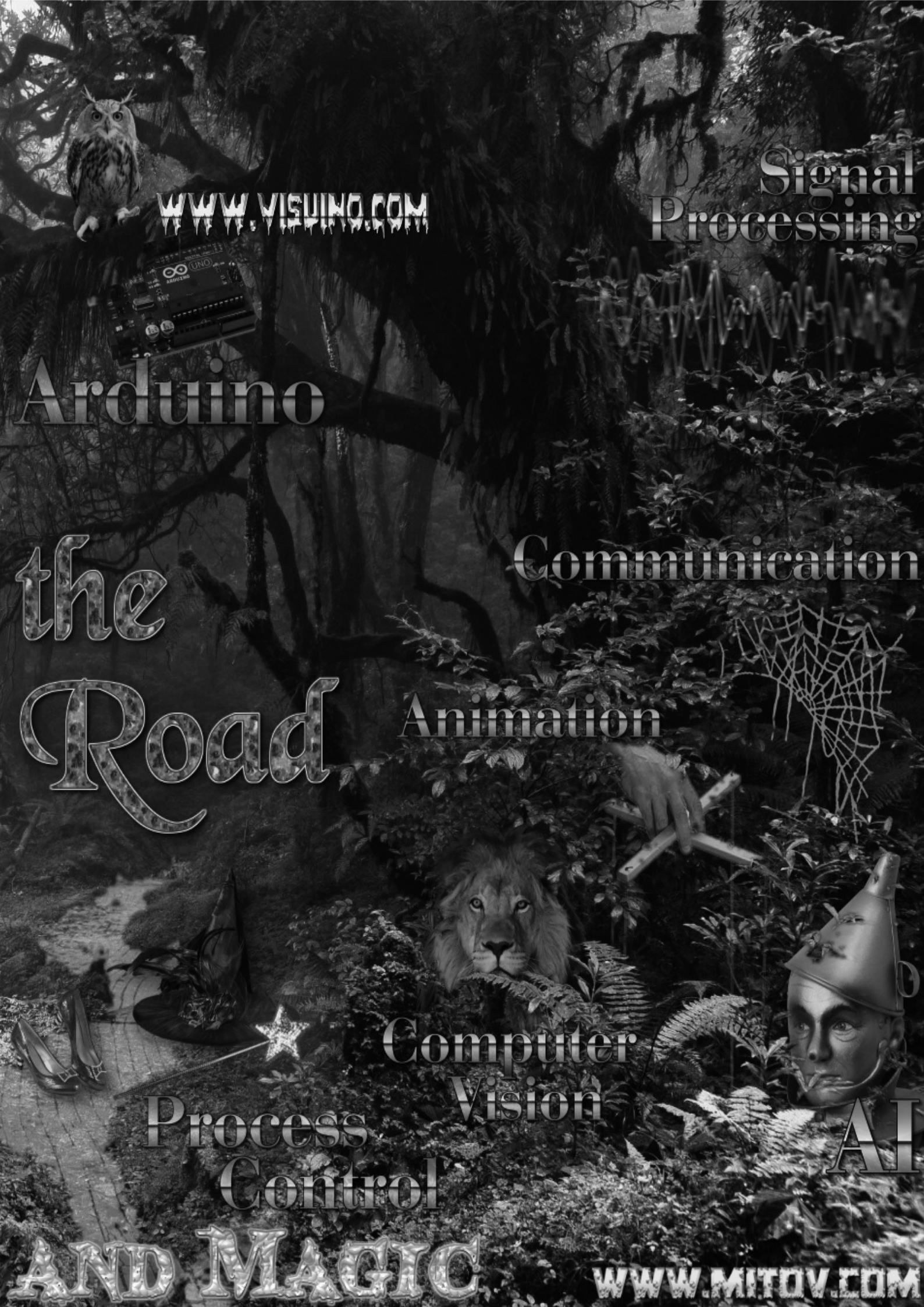
Follow  
Yellow Brick

Audio



TO THE WORLD  
OF TECHNOLOGY





[www.vividino.com](http://www.vividino.com)

Signal  
Processing

Arduino

Communication

the  
Road

Animation

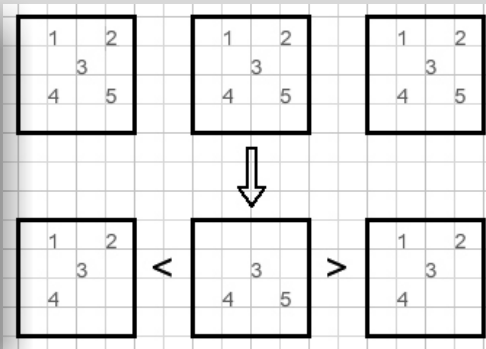
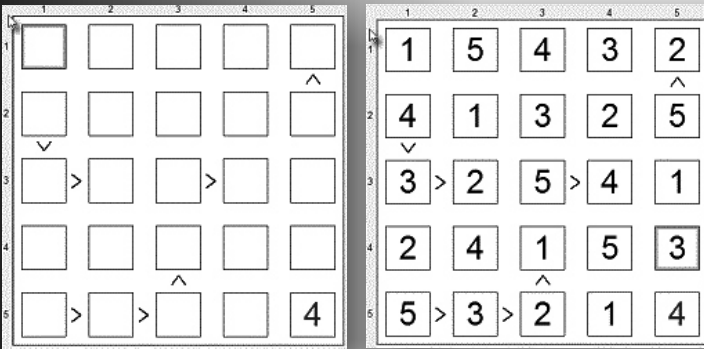
Computer  
Vision

Process  
Control

AI

AND MAGIC

[www.mitov.com](http://www.mitov.com)

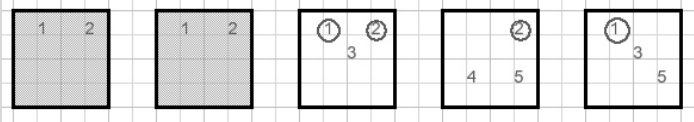


Above you see a futoshiki puzzle. Left the original- and right the solved puzzle. There are 25 fields of 5 rows and 5 columns. Per row/column the numbers 1..5 appear just once. Some numbers are filled in already. Between fields sometimes a < or > operator is placed indicating that adjacent number must be smaller or larger. The player has to apply logical deduction to find the number for each empty field

**1.** Below are five rules to solve futoshiki puzzles.

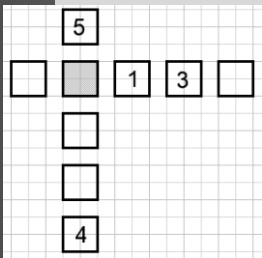
In the middle field the options 1,2 are removed. Also option 5 of the left and right fields are removed.

**5.** Look at the picture below and the field options:



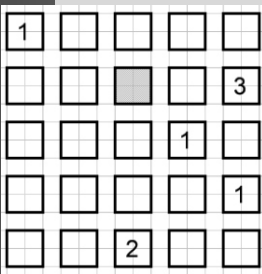
The left fields hold options 1,2. So these numbers cannot show up in the other fields. The removed options are placed in red circles.

**NOTE:** rules 1,2,5 are examples of one general rule: if in a row or column n fields hold n options, than these options cannot occur in the 5-n other fields. Rule 2. is the case where n = 4. The example at rule 5. is the case for n=2. This Delphi-7(and later) project assists in the solution of futoshiki puzzles.



Only number "2" fits in the orange field. I call this a "single option field".

**2.** Empty fields may hold some of the numbers 1..5, that do not occur in the other fields of the row or column. I call these numbers "options". In the image below the yellow field is the only field of column 3 that has the option "1" I call this situation a "single option field".

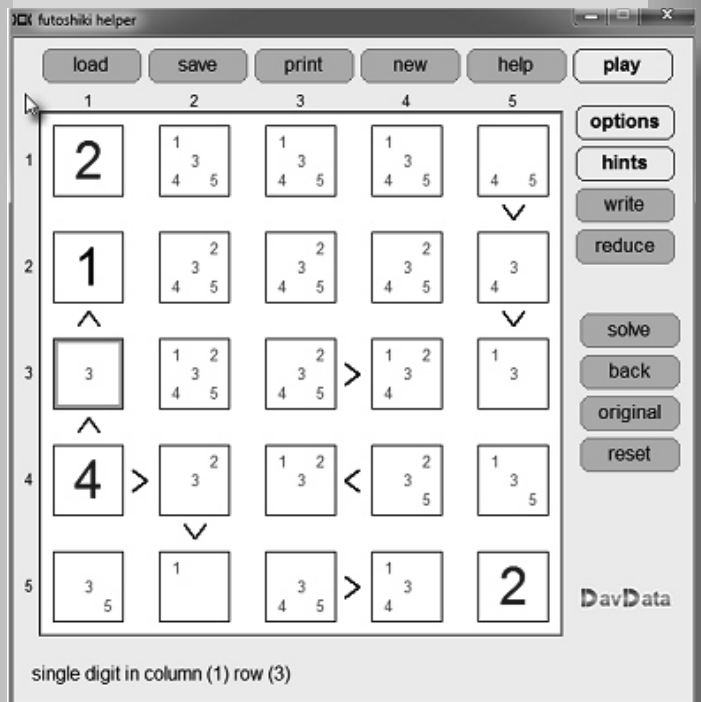
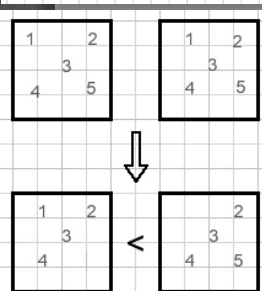


**3.** Presence of a < or > operator limits options.

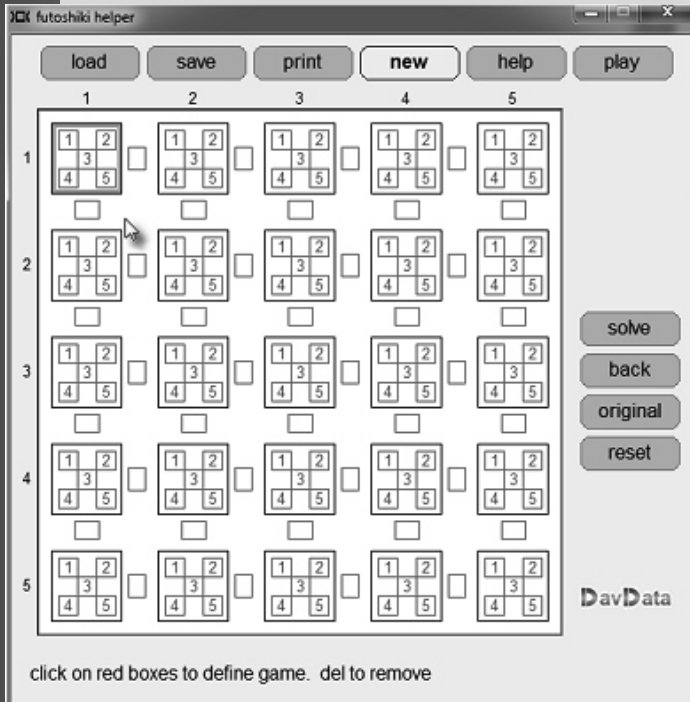
In the left field the option 5 is removed. In the right field this is option 1. Read right to left for the > operator.

In the left field the option 5 is removed. In the right field this is option 1. Read right to left for the > operator.

**4.** A field may be embedded between < > or > < operators



Menu buttons are placed horizontally at the top of the form.  
**Load** opens a previously saved puzzle.  
 File names have no extension but are prefixed with "fut-".  
**Save** saves a puzzle on disc.  
**Print** opens the printer dialog form.  
 This form may hold two puzzles. A checkbox allows for the printing of options.  
**New** Allows entry of a new puzzle.



An **orange** marker indicates the field where a number is added. The marker may be moved by the cursor buttons or the space bar.  
**Help** opens the internet help page.  
**Play** allows player to add numbers while searching for a solution.  
**Assistance.** The vertically placed buttons at the right side allow for assistance.  
**Options** Shows the options in each field.  
**Hints** Show hints such as the situation in rules 1,2 Give warning if field is out of options.  
**Write.** Performs the move indicated by the last hint.  
**Reduce.** Remove superfluent options in rows and columns.

**ASSISTANCE BUTTONS:**

**Solve.** Solves any puzzle by brute force method.  
**Back.** Takes last move back.  
**Original.** Puts puzzle in begin condition.  
**Reset.** Erases all fields.

**THE DELPHI PROJECT** There are 3 units  
 unit1 : handles events.

**Buttons** are of type davarrayBtn, part of dav7components, my own work.

**Paintbox** shows game.

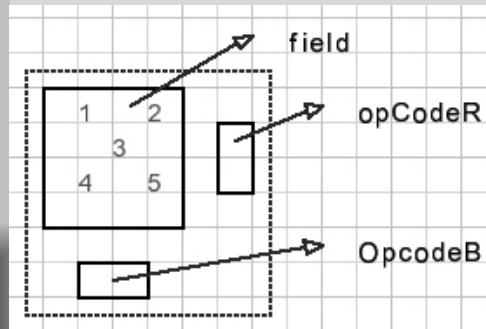
**Label** component is for messages.

**Game unit** : data structures and procedures for game

**print unit** : buttons and procedures to select printer and print games.

**DATA STRUCTURES**

A puzzle has 5 rows of 5 columns, so there are 25 fields.



```

type TFieldType = (mtFixed,mtPlayer,mtTrial);
Tfield = record
    bf : byte; //1,2,4,8,16,32
    ntype : TFieldType; //game; player; trial
    options : byte; //bits 1..5
    opcodeR : byte; //0:none;1<;2:> than right
    opcodeB : byte; //0:none;1<;2:> than bottom
end;
TMove = record
    mcol : byte;
    mrow : byte;
end;

var game : array[1..5,1..5] of Tfield;
    colOpts : array[1..5] of byte;
    rowOpts : array[1..5] of byte;
    movelist : array[1..25] of TMove;
    movecount : byte;
    
```

**ntype** may be:

**mtFixed:**

field contains a preset number, not changeable by player

**mtPlayer :**

this field number was typed by the player, not changeable by solver procedure

**mtTrial :**

the solver may add moves to search for a solution. The solver initially changes all empty fields to the **mtTrial** mode.

**bf (byte)**

holds the entered number in the following format:

- number 0 : byte = 0000 0001 (empty field)
- number 1 : byte = 0000 0010
- number 2 : byte = 0000 0100
- number 3 : byte = 0000 1000
- number 4 : byte = 0001 0000
- number 5 : byte = 0010 0000

The number is the bit set in the bf byte.

So, the next number n is (n shl 1)

Options : same as the bf field but a bit is set if the option is present.

**OpCodeR:**

- 0 : no opcode
- 1 : operator is <
- 2 : operator is >

**OpCodeB:**

similar as above for lower operator.

**ColOpts [1..5] :**

bit set if option is present in this column.

**RowOpts [1..5] :**

bit set if option is present in this row

**TMove :**

A player move is recorded as the column (**mcol**) and row (**mrow**) .

**MoveList[ . . . ]** is the list of player moves.

**MoveCount** is the number of player moves.

**CALCULATION OF OPTIONS**

In an empty puzzle all options are

\$3e = 0011 1110 (bits)

If a number (say 3) is added in column 1, row 5 then

- the fields bf value becomes 0000 1000 (bit 3 set).

- colOpts[1] becomes colOpts[1] xor bf

- rowOpts[5] becomes rowOpts[5] xor bf

A number entered in a field drops the options for that row and column.

**procedure CalculateOptions** takes care.

The presence of an operator further may drop options in an empty field. It does not affect row and column options. Things are a little more complicated here.

See procedure **procOperators** .

This procedure repeats itself until no more changes are noticed. Reason is, that operators may influence each other. There are several helper routines for the reduction of options by operators:

- **procOperatorField(..)**

- **HIMASK .....**

bitmask excluding the lowest option in a field

- **LOMASK .....**

bitmask excluding the highest option in a field

- **HIMASK2 .....**

bitmask excluding the 2 lowest options

( < .. > case)

- **LOMASK2 .....**

bitmask excluding the 2 highest options

(> .. < case)

The **LOMASK2** and **HIMASK2** functions have 1 byte as parameter input, which is the OR'd value of the options right and left of an operator.

See puzzle description for an explanation.

See the source code for details.

**MENU STRUCTURE**

The main menu is a **TDavArrayBtn** component having 1 row of 5 buttons.

One button may be down at the time.

The **Menubutton** variable holds the pressed button:

```
type TMenuButton =
```

```
(mbLoad,mbSave,mbprint,mbNew,mbHelp,mbPlay,mbOff);
```

```
...
```

```
var menubutton : TMenuButton
```

The value of **menubutton** directs events from keyboard or mouse to the proper procedures.

**NEW GAME**

**MenuButton = mbNew.**

There are two ways to enter a number in an empty field.

**1.By keyboard.**

Typed numbers are written in the field that is marked.

```
var markedRow : byte;
```

```
markedColumn : byte;
```

indicates where the marker is placed. The marker is moved by the cursor keys or the space bar.

**2.By mouse**

While moving the mouse pointer over the game, the area covered is recorded in variable **scanfield**.

```
type TscanType = (stNone,stOption,stOperator,stDigit);
Tscanfield = record
    stype : TscanType;
    scol : byte;
    srow : byte;
    snr : byte;
end;

...
var scanfield : Tscanfield;
...
function getScanfield(x,y : word) : TscanField;
begin
...
end;
```

**stype = stOption**

if mousepointer is over an option field.

**stype = stOperator**

if over an operator field.

**stype = stDigit** if over a number.

**scol** is the column,

**srow** is the row

**snr** is the option number or the operator field

(1 right, 2 down).

Function **GetScanfield** gathers this information during mouse moves.

On a **mouse down**, the scanfield information is processed.

**GENERATING HINTS**

When the Hint button is down (on) after a move the field options are searched for

1. a single option in a field

2. a field having the single option for it's row or column

2. a field being out of options

procedure **procHints**; takes care.

function **SingleBit(b : byte) : boolean**; is a helper, true is returned if byte b has just one bit set.

**REDUCING FIELD OPTIONS**

See point 5. of the game description.

This part is somewhat more complicated.

Observe a column or row having options per field, some fields filled in with a number. Finally, all fields are filled with numbers 1..5.

But not every number fits in a field.

Purpose is to squeeze out redundant options in each column and row.

All permutations are generated of numbers 1..5 and a permutation is compared against the field options. (*a permutation is a sequence of elements, numbers 1..5 in this case*). If there is a match (permutation allowed in row or column) than the bits per field are OR'd.

Say a row has options

(1,2) (1,2) (1,2,3,4) (1,2,3,5) (2,4,5)

Since the first 2 fields must hold numbers 1, 2 finally, these numbers cannot be options in fields 3,4,5.

Option reduction results in

(1,2) (1,2) (3,4) (3,5) (4,5)

At create time a table with all 120

permutations (0..119) of numbers 1..5 is generated.

procedure **makepermutations**; takes care

```
var permutation: array[1..5,0..119] of byte;
```

	1	2	3	4	5
0	1	2	3	4	5
1	1	2	3	5	4
119	5	4	3	2	1

```
function procPermGame: boolean;
checks the game rows and columns.
It calls function
procPermGroup(var grp:TGroup): boolean;
for each row and column.
```

**SOLVING A PUZZLE**

Press this button to solve a puzzle. The added numbers are colored orange. Press the solve button again to check if more solutions exist. A good puzzle however has one solution only.

```
function SolveGame(
  scode: TSolveCode): TSolveCode;
does the work.
```

Only the row- and column options are used, the options of individual fields are not used or altered.

variable m is the trial move (2,4,8,16,32) for numbers 1,2,3,4,5.

xcol, xrow are pointing to the field.

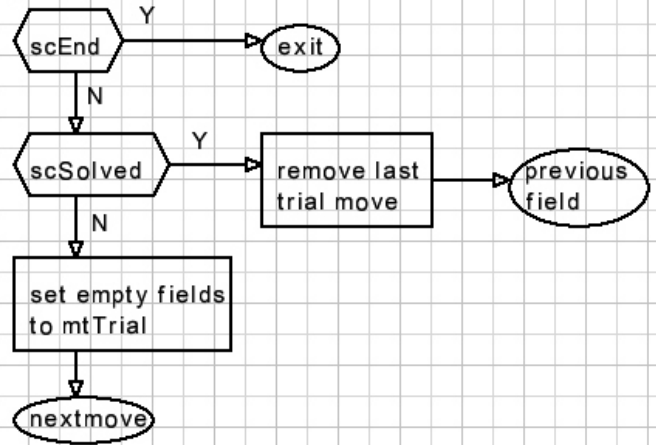
The method used is **Brute Force**.

Simply all numbers are tried in a systematical way until a solution is found or all possibilities are exhausted.

```
type TSolveCode = (scStart,scSolved,scEnd);
...
var solveResult: TSolveCode = scStart; //in unit1
```

```
scStart : unsolved game
scSolved : solution found (result) or search
for next solution (parameter)
scEnd : no solution found;
See flowchart below with entry part of function
SolveGame:
```

startup of Solve process



**Load and Save**

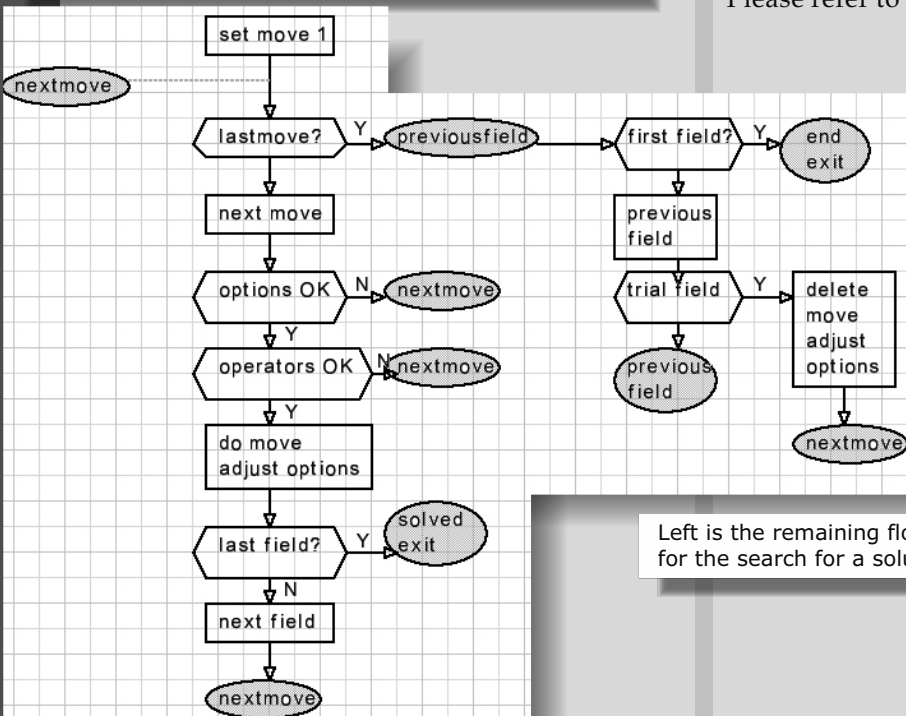
The save format is:

```
type TA = array[1..27] of word; // load,save
```

The game is saved as a typed file of TA.

```
word
1 | u | f |
2 | o | t |
3 | bf | x ntype R B | ---[1,1] field
.....
27 | bf | x ntype R B | ---[5,5] field
```

The filename is preceded by "fut-", there is no extension. This concludes the program description. Please refer to the source code for details.



Left is the remaining flowchart for the search for a solution.

## Trainingen

**Barnsten** organiseert in samenwerking met **Danny Wind** van de **Delphi Company** diverse Delphi en FireMonkey trainingen.

Het doel van de trainingen is u, als Delphi ontwikkelaar, snel productief te laten werken.

Er zijn zowel **Essentials als Advanced trainingen**

en daarmee aanbod voor zowel de beginnende als de ervaren Delphi ontwikkelaar.

Tijdens de trainingen maakt u gebruik van uw eigen laptop met daarop Delphi 10.2 Tokyo inclusief InterBase XE7 geïnstalleerd (*eventueel trial versie*).

13-15 november 2017

### **Delphi 10.2 Tokyo Essentials VCL Training**

Etten-Leur 1459,00

16-17 november 2017

### **Delphi 10.2 Tokyo Advanced Update Training**

Etten-Leur 995,00

starter expert **DX** Delphi

I have in the former two "REST easy with kbmMW" articles shown, how to make a REST server with kbmMW, and how to use that REST server to easily return and store data from/to a database all in less than 30 lines of real code.

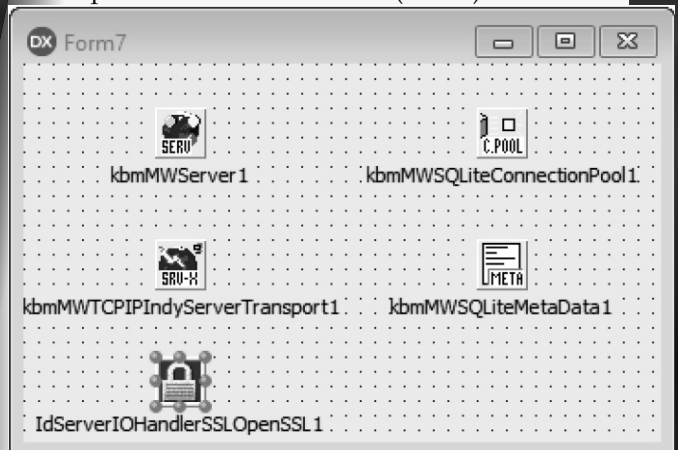
This article will center around how to ensure that communication with the server stays protected using SSL (Secure Socket Layer). In other words, how to make the REST server talk HTTPS rather than HTTP.

There are multiple ways to secure a kbmMW based application server with SSL, but I will focus on one simple way to do it using **OpenSSL**.

First we should create a certificate we can use. SSL certificates can be purchased from various places where they sell official certificates, or you can create one that is self signed. A self signed certificate is generally as secure as anything else, but it is not automatically trusted by other servers, which may flag your certificate as unsafe.

For inhouse use however, a self signed certificate is usually fine. There are many places on the internet explaining the procedure of how to create SSL certificates using OpenSSL. You can click here for one of them:  
<https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs>

Add a TIdServerIOHandlerSSLOpenSSL component to the main form (Unit7).



You will need to set it's SSLOptions properties like this:

**SSLOptions.Mode** must be **sslmServer**  
 Of the supported **SSLOptions.SSLVersions** I will suggest enabling only **sslvTLSv1\_2**

Leave the remaining properties as is at the moment.

Now double click the **OnGetPassword** event handler of the **IdServerIOHandlerSSLOpenSSL1** component to write some code in an event handler.

```
procedure
TForm7.IdServerIOHandlerSSLOpenSSL1GetPassword
(var Password: string);
begin
    Password:='yourCertificatePassword';
end;
```

The code going in the event should simply return the password you used when you created the private part of the certificate. It is required by OpenSSL to have access to this, to be able to use your private key.

Despite the above example, I would suggest you, **not to hardcode** the password inside your application, but rather **read it from an external configuration file**, of security reasons, in case your REST server executable got leaked elsewhere.

But for the current sample, with a homemade sample certificate, we can do with the hardcoded password.

# 3

Next we need to tell the kbmMW server side transport to let the OpenSSL code handle the main communication of our data. One part of that is to write an event handler for the OnConnect event of the kbmMWTCPIPIndyServerTransport1 component.

As you may notice, we change the port number for the first binding from 80 to 443, since we want to support the standard HTTPS port. You may also notice that it is possible to provide a Root certificate file. The Root certificate file typically contains a chain of public certificates that can be used by OpenSSL and browsers to verify that

```
procedure TForm7.kbmMWTCPIPIndyServerTransport1Connect(AContext: TIdContext);
begin
  if AContext.Connection.IOHandler is TIdSSLIOHandlerSocketBase then
    TIdSSLIOHandlerSocketBase(AContext.Connection.IOHandler).
      PassThrough:=false;
end;
```

To make the compiler happy, we also need to add **IdContext** to the interface sections uses clause.

your own certificate is a valid and trusted certificate generated by the entity from which you have the root certificate.

**uses**

```
Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
Vcl.Controls, Vcl.Forms, Vcl.Dialogs, kbmMWCustomTransport, kbmMWServer,
kbmMWTCPIPIndyServerTransport, kbmMWRESTTransStream,
kbmMWCustomConnectionPool, kbmMWCustomSQLMetaData, kbmMWSQLiteMetaData,
kbmMWSQLite, kbmMWORM, IdBaseComponent, IdComponent, IdServerIOHandler, IdSSL,
IdSSLOpenSSL, IdContext;
```

And finally we need to link the SSL component to the server transport, and make the certificate files available to OpenSSL.

Self signed certificates usually do not need any root certificate files.

We do that by writing the following piece of code, for example in the OnCreate event handler of the main form (containing the kbmMW Indy server transport).

Now your REST server is ready serving clients securely over SSL.

```
procedure TForm7.FormCreate(Sender: TObject);
begin
  ORM:=TkbmMWORM.Create(kbmMWSQLiteConnectionPool1);
  ORM.CreateTable(TContact);

  // Make sure that the server is now listening on port 443 which is
  // the default port for HTTPS communication.
  kbmMWTCPIPIndyServerTransport1.Bindings.Items[0].Port:=443;
  IdServerIOHandlerSSLOpenSSL1.SSLOptions.CertFile:='YourCertificateFile.cer';
  IdServerIOHandlerSSLOpenSSL1.SSLOptions.KeyFile:='YourPrivateKeyFile.key';

  // Optional root certificate file if purchased certificate or empty if self signed.
  IdServerIOHandlerSSLOpenSSL1.SSLOptions.RootCertFile:='';
  kbmMWTCPIPIndyServerTransport1.IdTCPServer.IOHandler:=IdServerIOHandlerSSLOpenSSL1;

  kbmMWServer1.AutoRegisterServices;
  kbmMWServer1.Active:=true;
end;
```

Make sure that the files YourCertificateFile.cer and YourPrivateKeyFile.key is available to the REST executable when it runs, but also make sure that they are not accessible for download for anyone else. It's of high importance that those files (along with your private key password) is kept a secret for anybody else.



starter expert DX Delphi

Building on the previous articles about how to create a REST server using kbmMW, we have now reached the stage where we should consider access management.

What is access management? It's the "science" of who are allowed to do what.

It is obvious that data exists in this world, which should be protected from being read, created or altered by people/processes we have not authorized to do so. Or turned on its head, some data should be protected and be accessible only by people/processes that we trust.

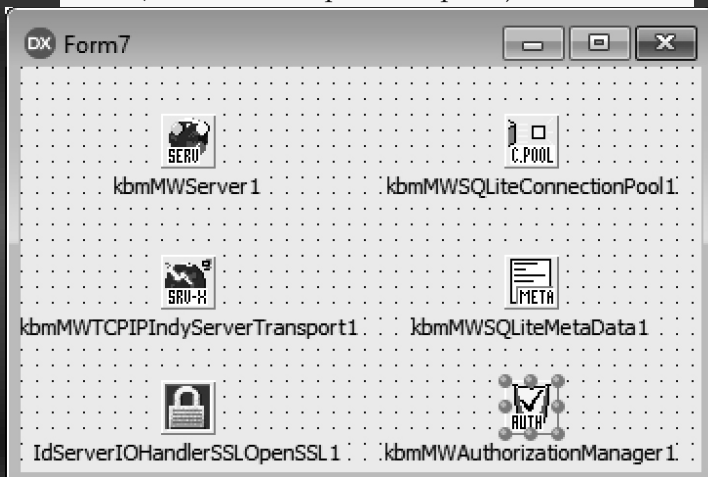
Other data might be left freely available for reading, but never for modifying and so forth.

Fortunately kbmMW have features built in to support us with that.

# 4

We start by adding a

`TkbmMWAuthorizationManager` to the main form (**Unit7** in the previous posts).



We can use the authorization manager as is, standalone, but it often makes sense to connect it to the kbmMWServer instance. Thus set the property

`kbmMWServer1.AuthorizationManager` to point on `kbmMWAuthorizationManager1`.

This way, every call into the application server will be checked by the authorization manager for access rights.

The kbmMW authorization manager is an entity which understands the topics:

- resource
- actor
- role
- authorization
- constraint
- login

A resource is basically anything that you want to add some sort of protection for.

It can be database related, it can be a specific object, it can be a function or a service that you want to ensure is only handled in ways that you want it to, by people/processes that you have granted access to it. Resources can be grouped in resource trees, where having access to one resource also automatically provides same access to resources underneath that resource.

An actor, is typically a person (*or a person's login credentials*), a process or something else that identifies "someone" that want access to your resource's.

A role is a way to categorize general access patterns. Roles in a library, could be a librarian, an administrator and a loaner. Roles in a bank could be a customer, a teller, a clerk, an administrator and so forth.

The idea is that each of the roles will have different access rights to the various resources. Actors usually will be given at least one role. An actor can have different roles, for example depending on how the actor logs in, or from where.

An authorization is a "license" to operate as an actor or a role on a specific resource. An authorization can be negative, thus specifically denying an actor or role access to specific resources and their subtrees.

A constraint is a limitation to an authorization or to a login. The authorization may only be valid within a specific timeframe, or be allowed to be accessed from specific equipment and such, or the login can only happen during daytime etc.

A login is the match between an actor/password and a login token. When an actor is attempting to be logged in, the system verifies login name, password, requested role and whatever constraints has been defined related to login in. Only when everything has been checked up and a login is allowed, a token is issued,

which the actor/user/process will need to send along with every request it makes to the kbmmw based server.

So let us define two roles we want to have access to our REST server. We can choose to name them 'Reader' and 'ReadWriter', but as kbmmw do not pose any restrictions to naming of roles (*nor on actors and resources*), we can name them anything as long as the names are unique within their category (*roles, actors, resources*).

- Reader
- ReadWriter

In code we define the roles like this (for example in the OnCreate event of the main form:

```
kbmMwAuthorizationManager1.AddRole('READER');
kbmMwAuthorizationManager1.AddRole('READWRITER');
```

We also, somehow, need to tell the authorization manager which actors exists so it can match up login attempts with actors.

The simple way is to predefine them to the authorization manager. That can for example also happen in the OnCreate event of the form, or elsewhere before the first access to the server. The actors can be defined from a database or a configuration file or LDAP etc. as needed.

```
kbmMwAuthorizationManager1.AddActor('HANS', 'HANSPASSWORD', 'READER');
kbmMwAuthorizationManager1.AddActor('CHRISTINE', 'CHRISTINEPASSWORD', 'READWRITER');
```

This defines two actors with their passwords, and which role they should act as upon login if they do not specifically ask for a different role.

It is possible not to predefine actors, but instead use an event handler to verify their existence in a different system via the OnLogin event of the kbmMwAuthorizationManager1 instance.

```
procedure TForm7.kbmMwAuthorizationManager1Login(Sender: TObject;
const AActorName, ARoleName: string; var APassPhrase: string;
var AActor: TkbmMwAuthorizationActor; var ARole: TkbmMwAuthorizationRole;
var AMessage: string);
begin
...
end;
```

An AActorName and the requested role name in ARoleName is provided. Optionally an actor instance may also be provided, if the actorname is known to kbmmw. If not, AActor is nil, and must be created by you if you know about the actor.

ARole may be nil, if it's an unknown role that is requested. You can choose to define the role on the fly by returning a newly created TkbmMwAuthorizationRole instance. Remember to add any newly created actor or role instances to the kbmMwAuthorizationManagers Actors and Roles list properties before returning.

APassword will contain the password delivered with the login attempt. You are allowed to modify it on the fly (*for example to change it to a SHA256 hash, so no human readable passwords are stored in the authorization manager*).

If you return nil for AActor or ARole, then it means that the login failed. You can provide an explanation in the AMessage argument if you want.

But let us continue with our simple actor definition for this sample.

Now that we have actors and roles defined, the authorization manager is ready to handle login attempts.

There is only one way to login, and that is by calling the Login method of the authorization manager. This can, for example, be called from a new REST function in your REST service.

An alternative is to let kbmmw automatically detect login attempts, and call the Login method for you. To do that, set the Options property of kbmMwAuthorizationManager1 to [mwaoAutoLogin].

As you may remember, all requests to the kbmmw server must be accompanied with a Token identifying a valid login. If that token is not available, kbmmw (*with mwaoAutoLogin set*), is triggered to use whatever username/password passed on from the caller, as data for a login attempt and will return the token back to the called if the login succeeded.

As a REST server is essentially a web server, adhering to the HTTP protocol standards, what happens when kbmMW detects an invalid (*or non existing*) login, is that kbmMW will raise an `EkbmMWAuthException`, which in turn (*when the call comes via the REST streamformat*), will be translated into an **HTTP error 401**, which is presented to the caller. In fact, if you would raise that exception anywhere within your business code and you do not manage it yourself, it will automatically be forwarded to the caller as a **401**.

This will prompt most browsers to present a login dialog, where username/password can be entered, and next call to back to the server, will include that login information. kbmMW will automatically detect this and use it.

So we have actor, role and login in place. Now we need to determine what resources we have. A resource can be anything you want to tag a unique name on.

Most of the time, it makes sense to define REST methods as a resource. This is done very easily in our smart service, where we have the functions for manipulating and retrieving contacts (Unit8). We use the `kbmMW_Auth` attribute.

What happens behind the scenes is that kbmMW automatically define resource names for the functions like this: `MyREST..AddContact`, `MyREST..GetContacts` etc.

**Notice the extra dot!** If we had defined the service to have a version, when we created it, that would be put between the dots.

As you can see, the resource name is just a string, and you can define all the resources you want to yourself, but know that if you use kbmMW smart services, it will automatically define resource names in the above format.

kbmMW will also automatically ask the authorization manager to validate that it is allowed to use a resource, upon a call from any client.

You can choose to make finer grained authorization by manually calling the authorization manager for validation of a call like this:

```
var res:TkbmMWAuthorizationStatus; sMessage:string;
begin
    ...
    res:=AuthorizationManager1.IsAuthorized(
        logintoken, 'YOURRESOURCENAME', sMessage);
```

```
[kbmMW_Service('name:MyREST, flags:[listed]')]
[kbmMW_Rest('path:/MyREST')]
TkbmMWCustomSmartService8 = class(TkbmMWCustomSmartService)
public
[kbmMW_Auth('role:[READER,READWRITER], grant:true')]
[kbmMW_Rest('method:get, path:helloworld, anonymousResult:true')]
[kbmMW_Method]
function HelloWorld:TMyResult;

[kbmMW_Auth('role:[READER,READWRITER], grant:true')]
[kbmMW_Rest('method:get, path:contacts, anonymousResult:true')]
function GetContacts:TObjectList;

[kbmMW_Auth('role:[READWRITER], grant:true')]
[kbmMW_Rest('method:put, path:addcontact')]
function AddContact([kbmMW_Rest('value: "{$name}"')] const AName:string;
[kbmMW_Rest('value: "{$address}"')] const AAddress:string;
[kbmMW_Rest('value: "{$zipcode}"')] const AZipCode:string;
[kbmMW_Rest('value: "{$city}"')] const ACity:string):string; overload;

[kbmMW_Auth('role:[READWRITER], grant:true')]
[kbmMW_Rest('method:get, path:"addcontact/{name}"')]
function AddContact([kbmMW_Rest('value: "{name}"')] const AName:string):string; overload;

[kbmMW_Auth('role:[READWRITER], grant:true')]
[kbmMW_Rest('method:delete, path:"contact/{id}"')]
function DeleteContact([kbmMW_Rest('value: "{id}"')] const AID:string):boolean;
end;
```

res can have the value of `mwasAuthorized`, `mwasNotAuthorized` OR `mwasConstrained`.

`mwasConstrained` means that the authorization might be given under different circumstances (*different time on day or similar*). The returned `sMessage` may explain in more detail what was the reason that the access was denied.

In a `kbmMW` smart service, you can get the login token (*logintoken*) as an argument to the method like this:

```
[kbmMW_Auth('role:[READER], grant:true')]
[kbmMW_Rest('method:get, path:"someCall"')]
function SomeCall([kbmMW_Arg(mwatToken)] const AToken:string):boolean;
```

When the `SomeCall` method is called, its `AToken` argument contains the `logintoken`.

You can also access the services `ClientIdentity.Token` property instead from within your methods if you do not want the token to be part of the argument list of your method call.

Now your REST server is protected by SSL and calls to its functionality limited by login. 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.

REST EASY WITH KBMMW PART 5  
LOGGING BY KIM MADSEN



Following up on the previous blog posts about how easily to create a REST server with `kbmMW`, I today want to write a little bit about logging.

`kbmMW` contains a quite sophisticated logging system, which lets the developer log various types of information whenever the developer needs it, and at runtime lets the administrator decide what type of log to react on and how.

In addition the log can be output in a file, in the system log (OS dependent), or be sent to a remote computer for storage. In fact all the above methods can coexist at once.

As you can tell, there seems to be various log requirements for various stages of the lifetime of the application:

- During development
- During usage
- Early warning
- Post incident investigation

A good log system should imo handle all the above scenarios, while making it simple to use for the developer, and allow the administrator to tune on the amount of information needed.

`kbmMW`'s log system handles all these scenarios, and can be late fine tuned for the required log level.

In addition the log system should be able to output the log in relevant formats, that match the application's purpose.

WHAT'S THE LOGGING?

Well. There can be multiple purposes, amongst

- For debugging while developing
- For debugging after deployment
- For keeping track of resources
- For keeping track of usage (perhaps even relates to later invoicing)
- For proving reasons for user complaints
- Of security reasons to track who is doing what

5

PURPOSE OF

Web server applications, might want to output some log data in a format generally accepted by web servers, and thus also by web server log file analyzer software, while other server applications may have other requirements for output.

**kbmMW** supports several output formats, and also allows adding additional formats, without having to make changes in the developer's logging statements. So let us get on with it.

First add the **kbmMWLog** **unit** to the units in which you expect to do some logging. In our case, we have the units **Unit7** (*main form unit*), **Unit8** (*Smart service unit... the actual REST business code*) and **Unit9** (*a defined sharable TContact object*).

It makes sense to add support for logging in **Unit7** and **Unit8**. In **Unit7** it would look similar to this:

**interface**

**uses**

```
Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes,
Vcl.Graphics,
Vcl.Controls, Vcl.Forms, Vcl.Dialogs, kbmMWCustomTransport, kbmMWServer,
kbmMWTCPiPIndyServerTransport, kbmMWRESTTransStream,
kbmMWCustomConnectionPool, kbmMWCustomSQLMetaData, kbmMWSQLiteMetaData,
kbmMWSQLite, kbmMWORDM, IdBaseComponent, IdComponent, IdServerIOHandler, IdSSL,
IdSSLOpenSSL, IdContext, kbmMWSecurity, kbmMWLog;
```

And in **Unit8** we have also added **kbmMWLog** to the uses clause. By simply adding this unit, we can already log by calling one of the methods of the public default available **Log** instance. Eg.

```
Log.Debug('some debug information');
Log.Info('2 + 2 = %d',[2+2]);
```

**kbmMW's** log system supports these easy access methods:

- **Debug**  
(typically used during development purposes),
- **Info**  
(inform about some non critical and non error like information)
- **Warn**  
(inform about some non critical anormal situation)
- **Error**  
(inform about some error, like an exception or something else which still allow the application to continue to operate)
- **Fatal**  
(inform about an error of such magnitude that the application no longer can run).
- **Audit**  
(inform about some information that you want to be used as evidence in a post analysis scenario).

They in turn calls a number of generic **TkbmMWLog.Log** method which takes arguments for log type, severity, timestamps and much more.

You can ask **kbmMW** to log content of streams, of memory buffers, XML and JSON documents, byte arrays, and you can even ask **kbmMW** to produce a stack trace along with your log (*not currently supported on NextGen platforms*).

In our simple REST server, we might want to log whenever a user logs in, when they are logged out, when a function is called, and when an exception happens.

To intercept the login situation, we will write some event handlers for the **OnLoginSuccess** and **OnLoginFailed** event on the **TkbmMWAuthorizationManager** instance we have on **Unit7**.

```
procedure TForm7.kbmMWAuthorizationManager1LoginFail(
Sender: TObject;
const AActorName, ARoleName, AMessage: string);
begin
Log.Warn(
'Failed login attempt as %s with role %s.%s'
,[AActorName,ARoleName,AMessage]);
end;
```

```
procedure
TForm7.kbmMWAuthorizationManager1LoginSuccess(
Sender: TObject;
const AActorName, ARoleName: string; const AActor:
TkbmMWAuthorizationActor;
const ARole: TkbmMWAuthorizationRole);
begin
Log.Info('Logged in as %s with role %s'
,[AActorName,ARoleName]);
end;
```

**It makes sense to log a successful login as an information, while an unsuccessful login is logged as a warning. If it happens often, it could be malicious login attempts, so warnings ought to be looked after.**

And we might also want to log a logout of a user. The logout may happen automatically due to the user being idle for too long. Refer to the previous articles for more information.

We might also want to log what calls are made by logged in users.

This can be done in many ways and many places. You could choose to do it within your business logic code in the smart service in **Unit 8**, which makes sense if you want to log some more specific information about the call.

But if you just want to log successful and failed calls, then it's easy to do so using the `OnServeResponse` event of the `TkbmMWServer` instance in **Unit 7**.

As long as the request is formatted correctly and thus served through the `TkbmMWServer`, it will be attempted to be executed, and a response sent back to the caller.

The execution may succeed or it may fail, but in all cases the `OnServeResponse` event will be triggered.

If you have **kbmMW Enterprise Edition** and thus also have access to the **WIB** (*Wide Information Bus*) publish/subscribe transports, you have a couple of additional log managers available for remote logging:

**TkbmMWClientLogManager**

- Publishes logs via the WIB

**TkbmMWServerLogManager**

- Subscribes for logs on the WIB, and forwards those through other log managers.

You can make your own log manager by descending from **TkbmMWCustomLogManager** and implementing the **IkbmMWLogManager** interface.

```
procedure TForm7.kbmMWServer1ServeResponse(Sender: TObject;
  OutStream: IkbmMWCustomResponseTransportStream; Service: TkbmMWCustomService;
  ClientIdent: TkbmMWClientIdentity; Args: TkbmMWVariantList);
begin
  if OutStream.IsOK then
    Log.Info('Successfully called %s on service %s',
      [ClientIdent.Func, ClientIdent.ServiceName])
  else
    Log.Error('An error "%s" happened while serving request for %s on %s',
      [ClientIdent.Func, ClientIdent.ServiceName, OutStream.StatusText]);
end;
```

Now we intercept and log at strategic places in our code, and in fact the logging is already working. But the log output is currently only placed on the system log, which on Windows is interpreted as the debugger.

We need to have our log output to a file, preferably with nice chunking when the file reaches a certain size.

The responsibility of the actual output, is the log manager. There are a number of log managers included with kbmMW:

**TkbmMWStreamLogManager**

- Sends log to a TStream descendant.

**TkbmMWLocalFileLogManager**

- Sends log to a file.

**TkbmMWSystemLogManager**

- Sends log to system log (depends on OS).

**TkbmMWStringsLogManager**

- Sends log to a TStrings descendant.

**TkbmMWProxyLogManager**

- Proxies log to another log manager.

**TkbmMWTEELogManager**

- Sends log to a number of other log managers.

**TkbmMWNullLogManager**

- Sends log to the bit graveyard.

To use a different log manager than the default system log manager, you simply create an instance of the log manager you want to use and assign it to the `TkbmMWLog.Log.LogManager` property. Eg.

```
Log.LogManager:=
TkbmMWLocalFileLogManager.Create(
'c:\temp\mylogfile.log');
```

However to set specific settings on the log manager, it is better to instantiate a variable with it, set its properties and then later assign that variable to the `Log.LogManager` property.

An even easier way, is to use one of the `Log.Output...` methods, which easily creates relevant log managers for you with settings that usually are good for most circumstances. Eg.

```
Log.OutputToDefaultAndFile('c:\temp\mylogfile.log');
```

This will in fact create 3 log managers, a system log manager, a file log manager and a tee log manager and automatically hooks them all up.

In our case we just want to output to a file, so let us stick with the `TkbmMWLocalFileLogManager`. So we will simply create an instance and assign it to the `Log.LogManager` as shown above.

Now all the log will be output to the file, and the file will automatically be backed up and a new created when it reaches 1MB size. Backup naming and size etc. are all configurable on the `TkbmMWLocalFileLogManager` instance.

You can control which fields are output via the `Log.LogManager.LogFormatter` property. It is default a `TkbmMWStandardLogFormatter`. `kbmMW` also supports a `TkbmMWSimpleLogFormatter` which only outputs date/time, type and the actual log string.

The standard log formatter also outputs data type, process and thread information and binary data (*usually converted to either Base64 or hexdump (pretty) format*).

There is much more to logging. We didn't touch the fact that the log system can handle separate log files for auditing and other logging, and that you can set filtering on each log manager so that particular log manager only logs certain log types or log levels or data types.

*Happy logging.*

## KBMMEMTABLE V. 7.77.20

### STANDARD AND PROFESSIONAL EDITION RELEASED!

**We are happy to announce the latest and greatest release of our memory table.**

#### Whats new in 7.77.10 September 16 2017

- **Added** support for SQL DDL statements: LIST TABLES, LIST INDEXES FOR TABLE xxx, DESCRIBE TABLE xxx, DESCRIBE INDEX xxx FOR TABLE xxx and some variations (ON instead of FOR, TABLE keyword optional in INDEX statement).
- **Added** support for CASE WHEN THEN ELSE END in both forms.
- **Added** support for NOT IN, NOT BETWEEN, NOT LIKE
- **Fixed** CREATE TABLE issues.
- **Added** support for SELECT INTO
- **Added** support for SQL multistatements. Statements separated by ; (semicolon)
- **Added** support for ALTER TABLE ADD COLUMN, ALTER TABLE DROP COLUMN, ALTER TABLE MODIFY /ALTER COLUMN, ALTER TABLE RENAME TO
- **Added** support for EXISTS TABLE and EXISTS INDEX
- **Added** support for DEFAULT value in CREATE TABLE
- **Added** support for UNIQUE constraint in CREATE TABLE
- **Improved** SQL field datatype parsing.
- **Added** support for OUT parameters in SQL custom functions.
- **Fixed** SQLReplace (Replace) incorrect argument index.
- **Added** SQLSplit (Split) custom SQL function to split strings.
- **Added** SQLRegExp (RegExp) custom SQL function for pattern matching and splitting
- **Added** SQLDataType (DataType) custom SQL function for splitting SQL datatype declaration.

- **Added** Options: `TkbmSQLOptions` property to `TkbmMemSQL`.  
`soOrderByNullFirst`
  - Default Null orders last in comparison`soOldFieldNamingSyntax`
  - Revert to old field naming syntax`soOldLikeSyntax`
  - Revert to old wildcard style like syntaxelse use true SQL style format using % and ?.
- **Added** multiple overloaded ExecSQL functions to `TkbmMemSQL` to allow easy one line calls. If source table names are not provided they will be named T1..Tn.
- **Changed** to support multiple SQL parsing errors before erroring out.

**Professional Edition is released with source and additional performance enhancement features to holders of an active kbmMW Pro/Ent Service and Update subscription (SAU).**

**A free CodeGear Edition can be found bundled with kbmMW CodeGear Edition for specific Delphi versions.**

`kbmMemTable` supports the following development environments:

- RAD Studio Delphi/C++ 10.2 Tokyo
- RAD Studio Delphi/C++ 10.1 Berlin
- RAD Studio Delphi/C++ 10 Seattle
- RAD Studio Delphi/C++ XE8
- RAD Studio Delphi/C++ XE7
- RAD Studio Delphi/C++ XE6
- RAD Studio Delphi/C++ XE5
- RAD Studio Delphi/C++ XE4
- RAD Studio Delphi/C++ XE3
- RAD Studio Delphi/C++ XE2
- Lazarus 1.2.4 with FPC 2.6.4



starter  expert



### Abstract

In this article we discuss the new FPReport reporting engine, the design goals that were at the basis of the engine, what can be done with it, and we show how it can be used.

### Introduction

Many applications need to print data from time to time. Delphi and Lazarus offer a printer canvas, on which you can paint the page to be printed, as if you would paint it on the screen. This works well for a small and quick print, but when printing needs become more complicated and elaborate, this approach is slow and cumbersome. For this reason, reporting tools exist. There are external reporting tools such as Crystal Reports, JasperReport. Integrated solutions for Delphi are FastReport, Quickreport, Rave reports and many others. Lazarus ships with lazReport (based on a free version of Fastreport 2), and FastReport has a version that compiles for Lazarus. They are banded reporting tools: the output is divided in 'bands' which are repeated once or more on a page. This can be a list of students attending a class, or an invoice for a customer purchase, or an quarterly overview of incoming funds... All these integrated engines share a common design fault: by design they require a GUI subsystem on the system where the report is generated.

Today, when more and more development is shifted to the web, this becomes an increasingly difficult restriction: many webservers are simple **Linux** containers without an **X-Windows** system installed.

While this requirement of having a **GUI** system present seems logical, it is not: strictly speaking, a generating a report is just layouting text and pictures on a page, determined by the data that drives the report. This is just a matter of calculating the sizes and positions of a series of rectangles on a virtual 'page'.

The **GUI** system can come in handy to design the report layout, and it is necessary to view the resulting output. But the actual layouting does not need a **GUI**.

A typical scenario is a webshop: the developers designs the invoice to be sent to the customer as a report, and integrates it in the web application: on the server the report design is stored (*it can be created in code in the binary, or as a file on disk*).

When the customer has finalized his purchase, a **PDF** is generated on the server, and sent to the client by mail.

Or the client can opt to show the invoice in the browser, in which case the report can be rendered to **HTML** directly. To generate this **PDF**, or the **HTML** based on the report design created by the programmer, no **GUI** system is needed.

**HTML** and **PDF** are just text files with layouting instructions - one for a browser, the other for a **PDF** reader. In this scenario, **PDF** and **HTML** are possible outputs of the reporting engine.

The manager in the office who wishes to see and print an overview the monthly purchases, may well be using a desktop program to access the web shop data. He can ask for a printed version of the monthly report. Here, no **PDF** is needed, the report can be sent directly to the printer or viewed on screen: again 2 forms of output for the reporting engine.

Considering all these use cases we arrive at a set of requirements:

- The core layouting engine may not rely on a **GUI** system to do its work. It results in a description of a set of output pages.
- Various output formats (*called renderers*) must exist: **PDF**, **HTML**, Image. These renderers again may not rely on a **GUI** system.
- Screen and printer are also output formats.
- The report designer to create the reports may depend on a GUI system.
- Multi-column layout must be possible.

These requirements are the basis for **FPReport**, with the **TFPReport** class as the main class. This set of requirements has an interesting consequence: To calculate the layout of a text, the reporting engine needs to know the extent of a text in the chosen font - a service that is commonly provided by the GUI subsystem, but which by our requirements, is unavailable. Luckily, the **freetype library** is a free library that can also provide this service for **TrueType** fonts. All classes in **FPReport** start with **TFPReport**, and each class **TFPReportNNN** is a simple descendant of a **TFPReportCustomNNN** class, the former publishes the **protected/** **published** properties of the latter.

**Important: In issue 64 there was a peace of code that was erroneously double placed in the list. Jou Now can find the right Code: HSBUTTON.zip**





**PRINTABLE ELEMENTS**

What should a reporting engine be able to print ? There are some obvious candidates, we call them report elements:

- **Text.** Preferably with some limited formatting inside the text: bold, colors, and for PDF or HTML output: hyperlinks. The text should be customizable: this means that we must be able to get it from a data source, and we should have some formatting options available.  
(the class **TFPReportMemo**)
- **Images.** This can be a company logo, but can also be an image of an item you purchased, or the picture of a student in a list of students..  
(the class **TFPReportImage**) . It can load any **FPC** supported image type.
- **Checkboxes:** these are just a special case of an image: an image to represent 'true', and an image to represent false.  
(**TFPReportCheckbox**)
- **Shapes:** squares, circles, triangles or simple lines. (**TFPReportShape**)

But preferably the list of 'printable' things should be larger:

- **Barcodes**  
(available in **TFPReportBarcode**) .
- **QR codes**  
(available in **TFPReportQRCode**) .
- **Graphs.** (not yet available, but planned)
- **Pivot tables.** (not yet available, but planned)
- ...

The system must be extensible: it must be possible to register additional printing elements, and a renderer for this element must exist. The most common renderer simply draws whatever is needed on a bitmap, and then the report renderer draws the bitmap on screen, in **HTML** or whatever output is desired.

It is possible to create and register renderers for a specific format (for improved quality of output), but this is entirely optional.

**FPReport** comes with barcode and **QRCode** renderers.

It allows to use simple **HTML** tags inside text elements (*bold, italic, anchor, font*), and allows you to embed formulas in the text.

**DATA AND CALCULATIONS**

Formula in the text will be replaced by their calculated result in the output. The reporting engine uses the **Free Pascal** expression parser engine to provide formula support. This engine allows the use of variables (*identifiers*), meaning that you can do something like

The amount to be paid is  
[formatfloat('##0.##',total)] EUR.

The text between square brackets is a formula, which will format the variable 'total' using the `formatfloat` function. All fields in the data of the report is available as variables in the formula. It is also possible to add named report variables to a report: the value of these variables will be made available in the formula by their names. The engine also supports aggregate data such as `Min(SomeVariable)`

The total amount is  
[formatfloat('##0.##',sum(itemprice))] EUR.

If `itemprice` represents the price of an item in the invoice, the engine will update the formula with each iteration over the items in the invoice.

A report is driven by data. Traditionally this data comes from a database, and is fetched through a dataset: The report loops over the records in the dataset, and prints a detail band for each record in the dataset.

The idea of looping over data can be generalized, and **FPReport** supports several 'data loops' (*all descendents of TFPReportDataLoop*) out of the box:

- **A dataset-backed loop.**  
(**TFPReportDatasetData**) .

Just hook up the dataset to the report. This can be done visually in an IDE, there is no need to create code.

- **A JSON-array backed loop.**  
(**TFPReportJSONData**)

The array contains objects, and each property of the object is available as a field.

- **A Collection backed loop**  
(**TFPReportCollectionData**) .

The published properties of the collection items are the data available in the report.

- **A list backed loop.** The published properties of the objects in the list are the data available in the report. (**TFPReportListData**) .

- **A user event driven loop**  
(**TFPReportUserData**) :

if none of the above suits your needs, a simple solution is to use the event driven data loop: here the names and values of variables are fetched through events, and when the loop needs to go to the next iteration of the loop, it calls an event as well.

These loops are implemented in separate units, so the only code that you actually use is included in your application. This means is possible to create reports without including any database code in your application.



### CREATING A REPORT IN CODE

To get a feel for what is involved in designing and a report, we'll create a report in code. It's a simple report, it just prints the contents of a text file, nicely formatted. It adds a page header with date and filename, and a page footer with the page number. Instead of loading the `stringlist` contents from file, this could be the contents of a memo: the code can be used to print the contents of a memo instead of a file. The program is extremely simple, the main code is in the `DoRun` method.

```
procedure TPrintApplication.DoRun;
Var
PG : TFPReportPage;
PH : TFPReportPageHeaderBand;
PF : TFPReportPageFooterBand;
DB : TFPReportDataBand;
M : TFPReportMemo;
PDF : TFPReportExportPDF;
Fnt : String;
begin
Fnt:='DejaVuSans';
FLines.LoadFromFile(ParamStr(1));
gTTFontCache.ReadStandardFonts;
gTTFontCache.BuildFontCache;
PaperManager.RegisterStandardSizes;
```

The first two lines speak for themselves. The `ReadStandardFonts` and `BuildFontCache` lines tell the font engine to load standard fonts from standard locations.

This is a `catchall` method, which registers all available fonts. More fine-grained control is possible. The important thing is that the engine loads in memory the needed font information before the reporting engine starts laying out the report.

After that the `RegisterStandardSizes` call is used to register a set of commonly used page sizes. Again, this is necessary once, to be able to set the paper size of a report page. The next step is adding a design page to the report:

```
// Page
PG:=TFPReportPage.Create(FReport);
PG.Data:=FData;
PG.Orientation:=poPortrait;
PG.PageSize.PaperName:='A4';
PG.Margins.Left:=15;
PG.Margins.Top:=15;
PG.Margins.Right:=15;
PG.Margins.Bottom:=15;
```

If no paper size is set, then unexpected things can and will happen. Setting the margins is natural, the whole page cannot be filled by a printer. The used units are millimeters. Note that the page owner is the report.

This is not a requirement, but doing so adds the page to the report: a report can have multiple designer pages, which will be rendered one after the other.

**NOTE** that the page data is set to `FData` - this is an event data loop, which will be set up later. The reporting engine needs to know for each design page which data loop must be run. Once the page is set up, we set up a page header with 2 memos: one to contain the filename of the printed file, the other contains the date:

```
// Page header
PH:=TFPReportPageHeaderBand.Create(PG);
PH.Layout.Height:=10;//1 cm.
// Filename
M:=TFPReportMemo.Create(PH);
M.Layout.Top:=1;
M.Layout.Left:=1;
M.Layout.Width:=120;
M.Layout.Height:=7;
M.Text:=ParamStr(1);
M.Font.Name:=Fnt;
M.Font.Size:=10;
// date
M:=TFPReportMemo.Create(PH);
M.Layout.Top:=1;
M.Layout.Left:=PG.Layout.Width-41;
M.Layout.Width:=40;
M.Layout.Height:=7;
M.Text:='[Date]';
M.Font.Name:=Fnt;
M.Font.Size:=10;
```

The filename is just entered as the contents of the report memo. The date is entered using a formula: the `Date` function is available in formulas used in the report, and will be formatted using standard date notation (*obviously, there are functions to change the formatting*).

Similarly, we can set up the page footer:

```
// Page footer
PF:=TFPReportPageFooterBand.Create(PG);
PF.Layout.Height:=10;//1 cm.
M:=TFPReportMemo.Create(PF);
M.Layout.Top:=1;
M.Layout.Left:=1;
M.Layout.Width:=40;
M.Layout.Height:=7;
M.Text:='Page [PageNo]';
M.Font.Name:=Fnt;
M.Font.Size:=10;
```

The `PageNo` variable contains the current page. The `PageCount` variable is also available, and contains the total number of rendered pages. The page count can be substituted at the end of the rendering, or the report can be rendered twice (*this happens automatically*) and will be set to the number of pages that were rendered at the end of the first run.



All that must be done is create a band in which the contents of the string list will be displayed. The data loop will return 1 line of the string list on each iteration. That means that the data band will be printed once for each line in the string list. So we set up a data band with 1 memo that stretches over the width of the band:

```
// Actual line
DB:=TFPReportDataBand.Create(PG);
DB.Data:=FData;
DB.Layout.Height:=5;//0.5 cm.
DB.StretchMode:=smActualHeight;
M:=TFPReportMemo.Create(DB);
M.Layout.Top:=1;
M.Layout.Left:=1;
M.Layout.Width:=PG.Layout.Width-41;
M.Layout.Height:=4;
M.Text:=[Line];
M.StretchMode:=smActualHeight;
M.Font.Name:=Fnt;
M.Font.Size:=10;
```

The memo contains a formula with a simple variable name: Line. This is the name that our data loop will need to report. The memo by default will perform wordwrap, and we set the stretch mode of the memo and the band to **smActualHeight**. This means that the memo will increase its height to fit the length of the text, so long lines are accommodated. In turn, the band will increase its height as the memo grows in height. By default, the heights of bands and printable elements is fixed (**smDontStretch**). Then we set up our data loop, which is event based:

```
// Set up data
FData.OnGetNames:=@DoGetNames;
FData.OnNext:=@DoGetNext;
FData.OnGetValue:=@DoGetValue;
FData.OnGetEOF:=@DoGetEOF;
FData.OnFirst:=@DoFirst;
```

The events are extremely simple. The first one reports the 'variables' that are managed by the loop, as we've seen the memo expects a 'Line' variable:

```
procedure TPrintApplication.DoGetNames(Sender: TObject; List: TStrings);
begin
List.Add('Line');
end;
```

As the report loops over the data loop, the next line in the stringlist must be returned. In essence this loop is coded as

```
Data.First;
While not Data.EOF do
begin
// Get data and Print bands
Data.Next;
end;
```

So we need a current line index, we'll keep it in a **FLineIndex** variable which is initialized, updated and checked in the following routines:

```
procedure TPrintApplication.DoFirst(Sender: TObject);
begin
FLineIndex:=0;
end;
procedure TPrintApplication.DoGetNext(Sender: TObject);
begin
Inc(FLineIndex);
end;
procedure TPrintApplication.DoGetEOF(Sender: TObject;
var IsEOF: boolean);
begin
IsEOF:=FLineIndex>=FLines.Count;
end;
```

Finally, when converting the memo formula to text, the **OnGetValue** event is called, and it needs to return the correct variable. There is only one variable, so this is easy:

```
procedure TPrintApplication.DoGetValue(Sender: TObject;
const AValueName: string;
var AValue: variant);
begin
AValue:=FLines[FLineIndex];
end;
```

In an actual report with more variables, a check on **AValueName** would have to be performed, and the correct value corresponding to the name would have to be returned. Now everything is set up and the report can be rendered:

```
// Go !
FReport.RunReport;
PDF:=TFPReportExportPDF.Create(Self);
try
PDF.FileName:=ChangeFileExt(Paramstr(1),'.pdf');
FReport.RenderReport(PDF);
finally
PDF.Free;
end;
```

The **RunReport** method does the actual layouting of the report. The result of this layouting is a structure in memory which can then be rendered. The rendering happens using the appropriate rendering class, and in the example above, we render the report to a **PDF** File using the **TFPReportExportPDF** class. The result of this is a **PDF** file, which can look as in figure 1 on the next page.



```

txt2pdf.pdf
File Edit View Go Bookmarks Help
Previous Next 1 of 4 70%
14TH OCTOBER 2017
LAST REMINDER
YOU NEED TO REGISTER
PASCON FOR LAZARUS
txt2pdf.pas
program txt2pdf;

{$mode objfpc}{$H+}

uses
  Classes, SysUtils, CustApp, fpreport, fpreportpdfexport, fpTTF;

type

  { TPrintApplication }

  TPrintApplication = class(TCustomApplication)
  private
    FReport : TFPReport;
    FLines : TStringList;
    FData : TFPReportUserData;
    FLineIndex : Integer;
  procedure DoFirst(Sender: TObject);
  procedure DoGetEOF(Sender: TObject; var IsEOF: boolean);
  procedure DoGetNames(Sender: TObject; List: TStringList);
  procedure DoGetNext(Sender: TObject);
  procedure DoGetValue(Sender: TObject; const AValueName: string; var
  AValue: variant);
  protected
    procedure DoRun; override;
  public
    Constructor Create(AOwner : TComponent); override;
    Destructor destroy; override;
  end;

  { TPrintApplication }

  procedure TPrintApplication.DoGetNames(Sender: TObject; List: TStringList);
  begin
    List.Add('Line');
  end;

  procedure TPrintApplication.DoGetEOF(Sender: TObject; var IsEOF: boolean);
  begin
    IsEOF := FLineIndex >= FLines.Count;
  end;

  procedure TPrintApplication.DoFirst(Sender: TObject);

```

Figure 1: The program runs on its own source code

**Conclusion**

The above is just a small example of what FPReport can do. In a next article, we'll show how reports can be designed visually, and how to load a report design from file. We'll also discuss more advanced grouping and Master-detail relations.



## INTRODUCTION:

This is a very simple to use article about installing Virtual Box and Linux Mint 1.8 and Lazarus 1.8. The installation is very easy and cost almost no time at all. All programs are for free. Here we explain the installation order, the most necessary parts you need to know – having in mind we want to install Lazarus 1.8 under Linux Mint 1.8.

### VirtualBox is a x86 and AMD64/Intel64 virtualisation product for enterprise as well as home use.

VirtualBox and Lazarus are solutions that are **freely available as Open Source Software** under the terms of the GNU General Public License (GPL) version 2.

**Linux Mint is free of charge** and we hope you'll enjoy it. Some of the packages we distribute are under the GPL. Linux Mint is copyrighted 2006 and trademarked through the Linux Mark Institute.

### Lazarus is cross-platform IDE for Free Pascal.

Free Pascal is a GPL'ed compiler that runs on Linux, Win32, OS/2, 68K and more. Free Pascal is designed to be able to understand and compile Delphi syntax, which is OOP.

Lazarus is the part of the missing puzzle that will allow you to develop Delphi like programs in all of the above platforms. **Lazarus and Free Pascal** strives for write once compile anywhere. Since the exact same compiler is available on all of the above platforms it means you don't need to do any recoding to produce identical products for different platforms.

It is well-known that making use of virtualisations is a very important way of organizing your computer life. It helps in the way that you can use many Operating Systems and environments for using Delphi or Lazarus and develop.

A very good extra is that you can Import and Export VDI's (*these files contain all your instalment data*) so that you can reuse them or use them as Backup.

In a separate article I will explain Virtualisations in depth.

## WHERE TO GET VIRTUAL BOX

<https://www.virtualbox.org/>

paths on your local Drive:

c:\Users\Detlef\VirtualBox Vms\  
Mint 1\_8

## WHERE TO GET LINUX MINT

<https://linuxmint.com/download.php>

## WHERE TO GET LAZARUS

<http://www.lazarus-ide.org/index.php?page=downloads>

or

<https://sourceforge.net/projects/lazarus/files/>

Lazarus%20Linux%20amd64%20DEB/

Lazarus%201.8.0RC4/

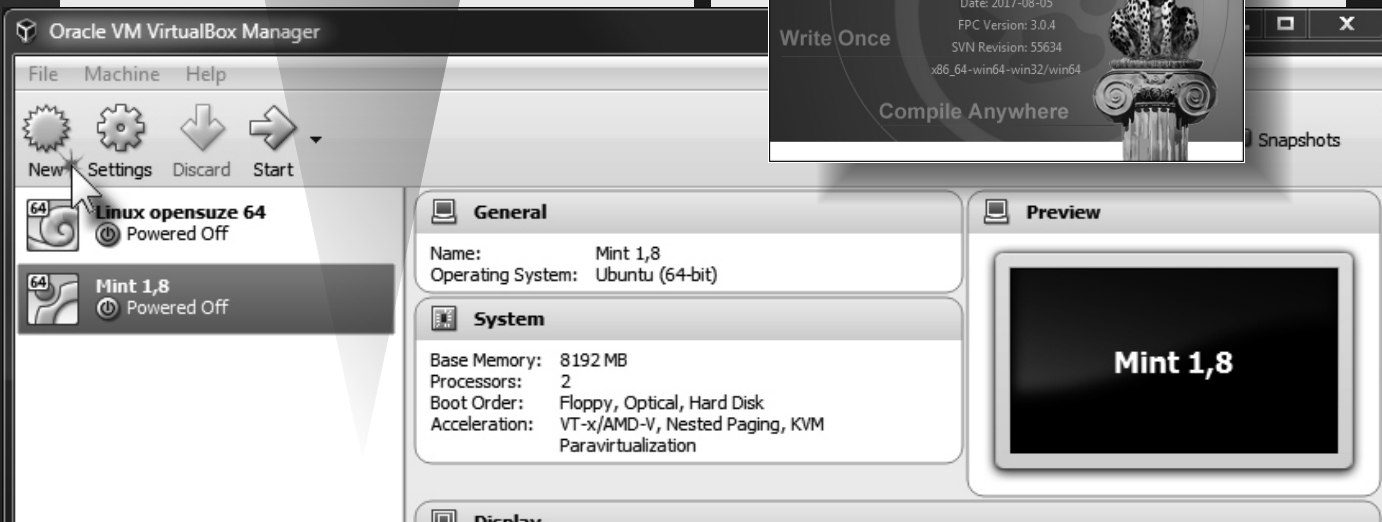
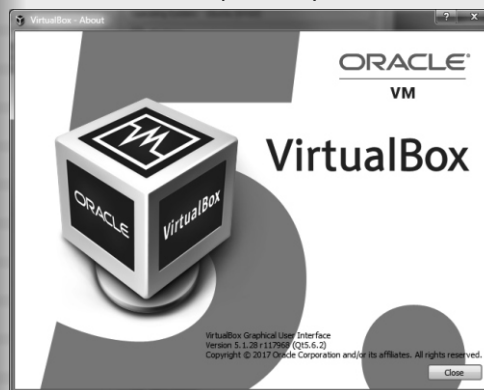
You need to download all three Lazarus files!

The installment we used are version Lazarus

1.8 Release Candidate which means the

latest version available. Stable versions will

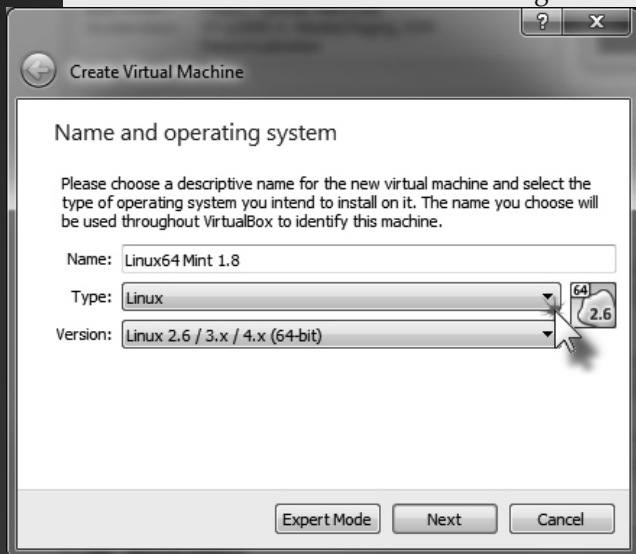
be announced separately.





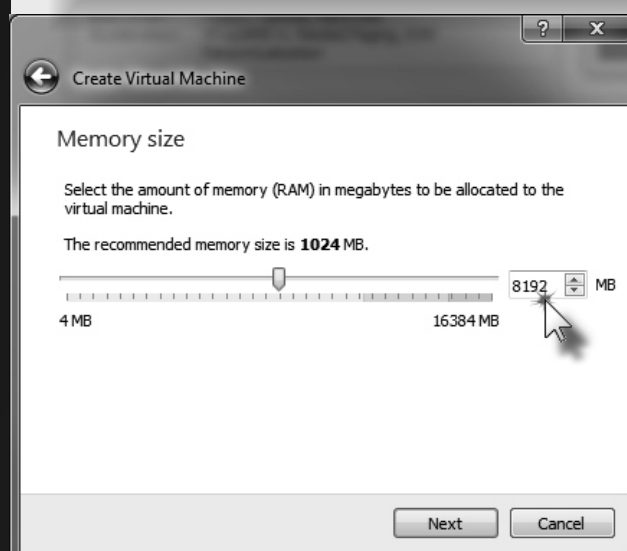
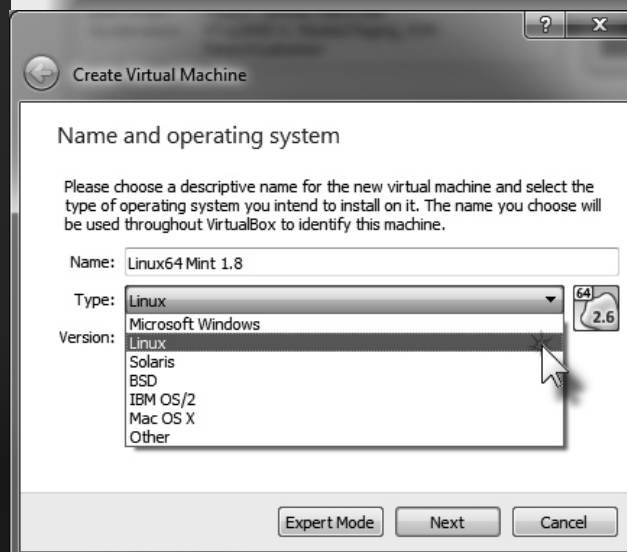
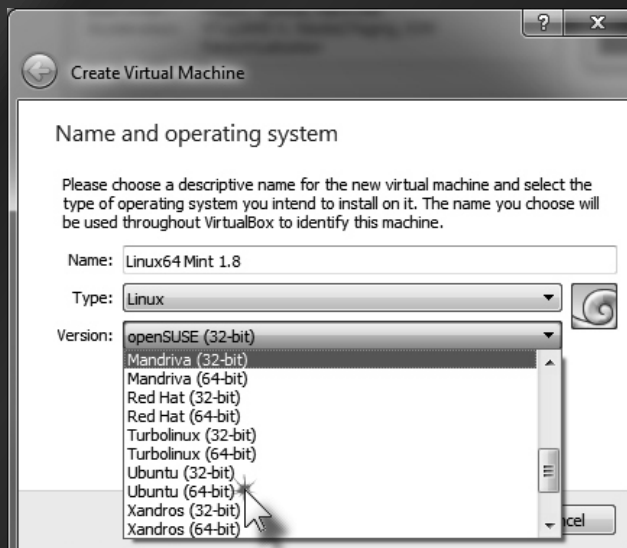
So after mentioning all the rights and make sure you have the right addresses to start this project, we need to begin with downloading VirtualBox and start installing. If you need to find where the material is placed on your hard disk it will be probably: `c:\Users\Detlef\VirtualBox Vms\Mint 1_8` under win 7 and 10.

I installed it and tested it without problems. Mint 1\_8 is the name I gave to the project, so if you have another name that will change of course to that name as well. I will show as much information to you to make it all very easy to follow the instructions. The first window you will see if you click on the NEW button is the next Figure.

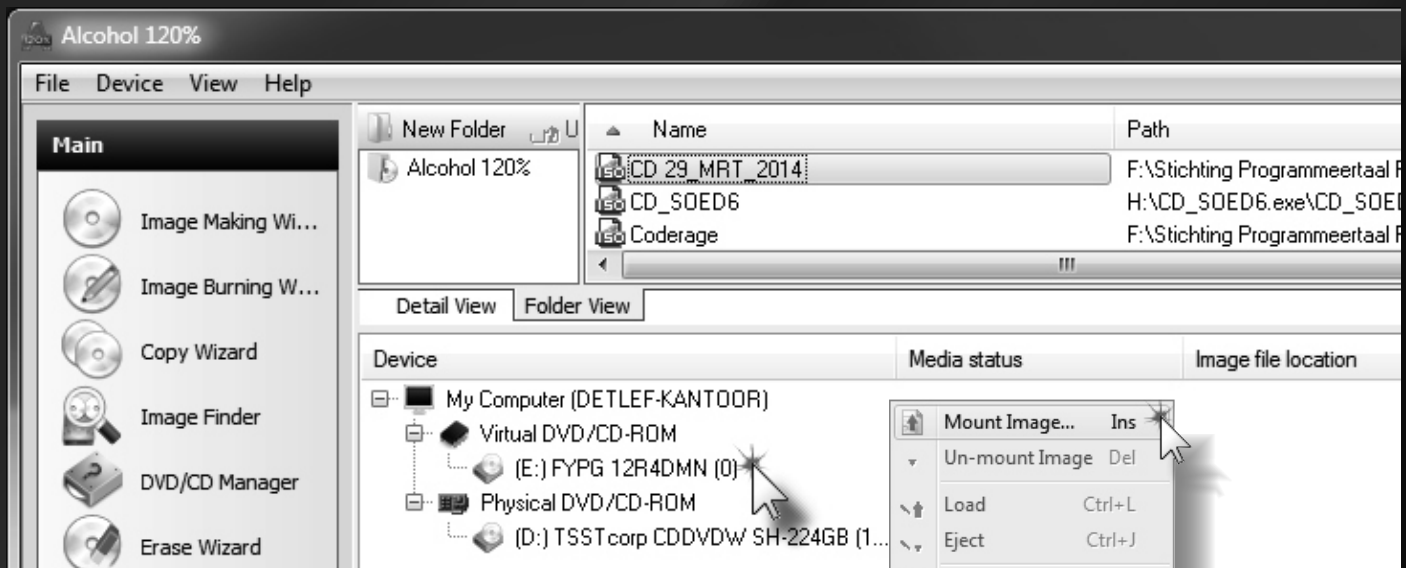


The name is what ever name you have in mind, probably use a description of the project. You can change it at any time. You can also change the language of virtual box:

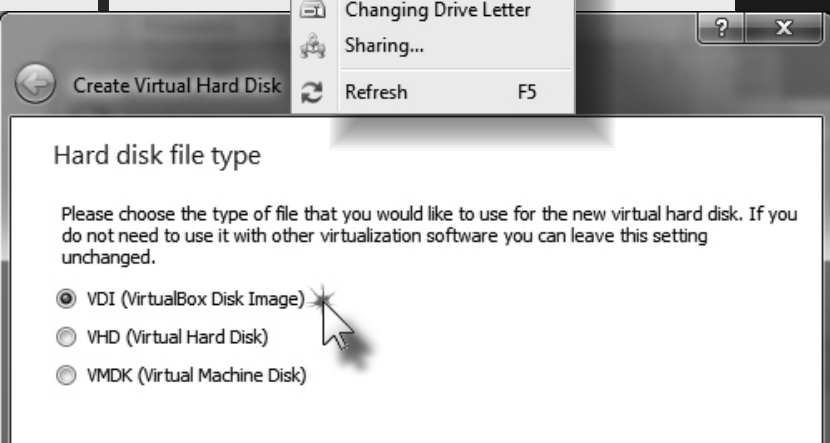
-> **File|Preferences|Language.** The Type should be aiming at the sort of OS you might use. You could make choice by using the drop down list. The version will also change and you can make choice there or type in something that you have in mind. It has no actual consequences.

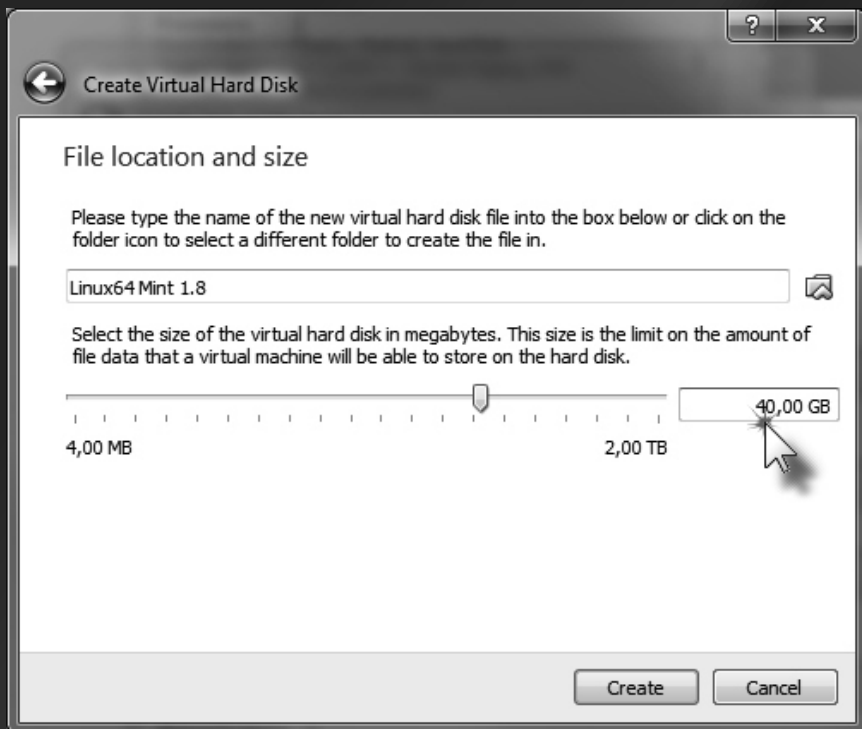


Now speaking about Memory size: it always is better to have a lot then to have just what you need. It makes the systems run much faster.



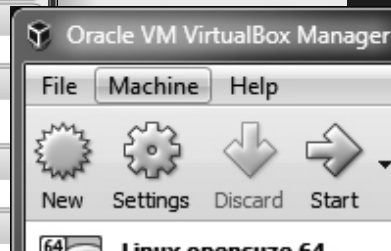
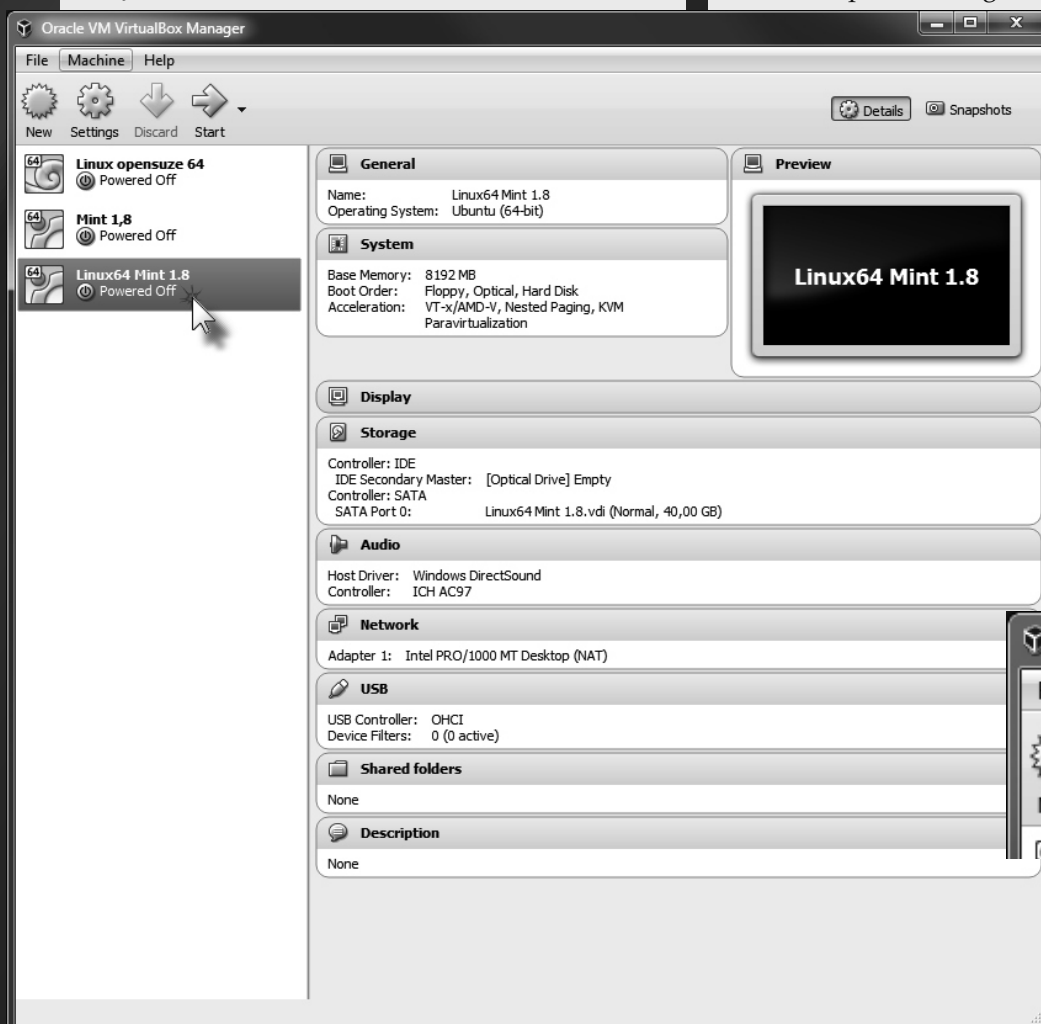
Before you can start creating the virtual hard disk you really need to organize a few things first:  
Here is what I did: because I use win7 in this case I Downloaded an ISO from Linux Mint.  
This I dropped in to my version of Alcohol which is very good for these purposes: create your extra virtual DVD and place the ISO into the Virtual DVD. Virtual Box accepts that as a source for loading your OS.  
Now there are of course other possibilities: you could (In win 8/10) use the ISO reading option, or if you can't burn the ISO to disk and use it. You also can create your own Virtual disk and then read it and copy the content to a special directory.





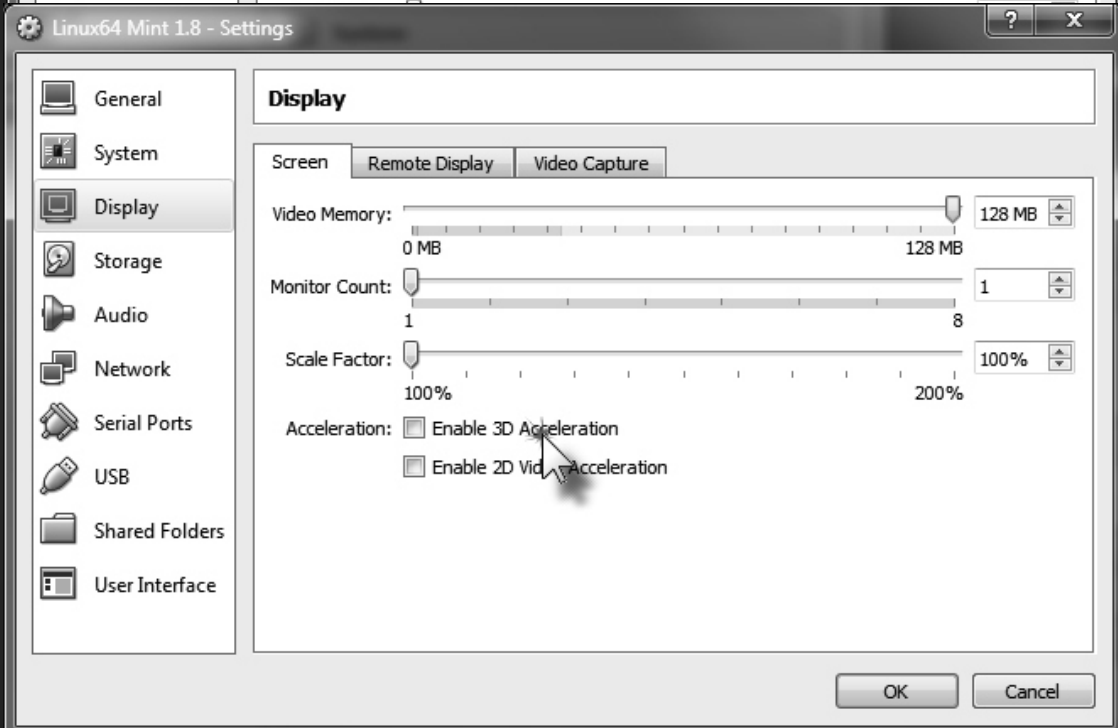
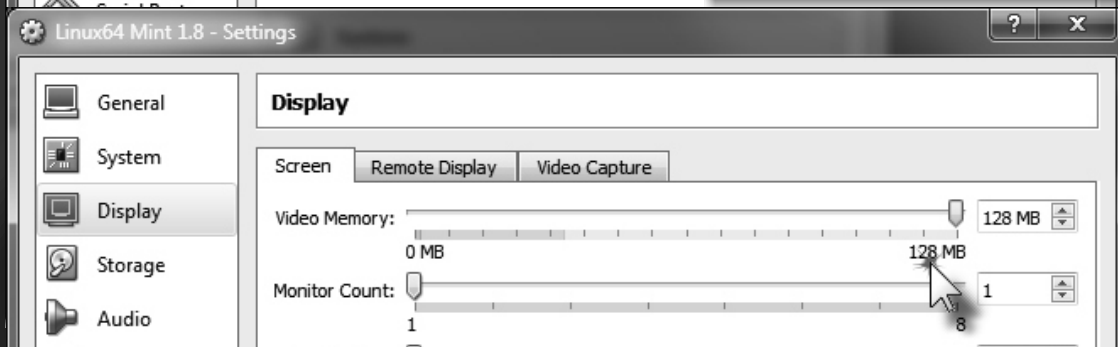
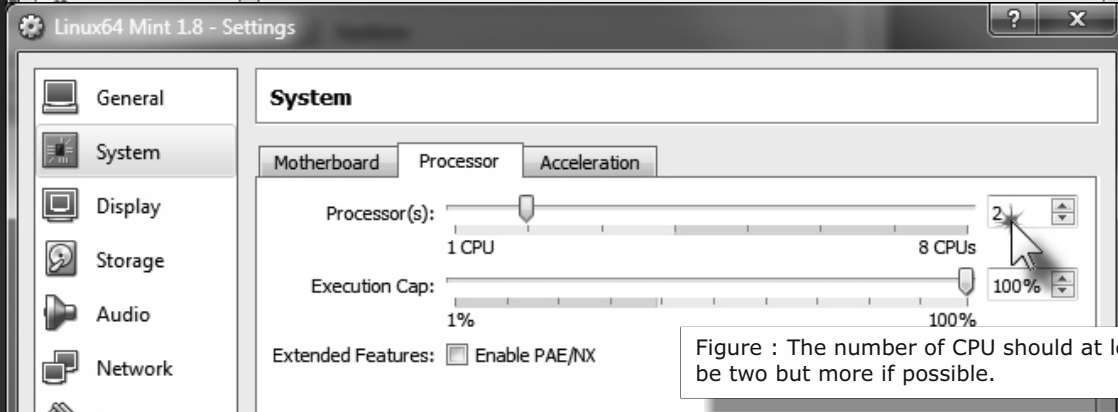
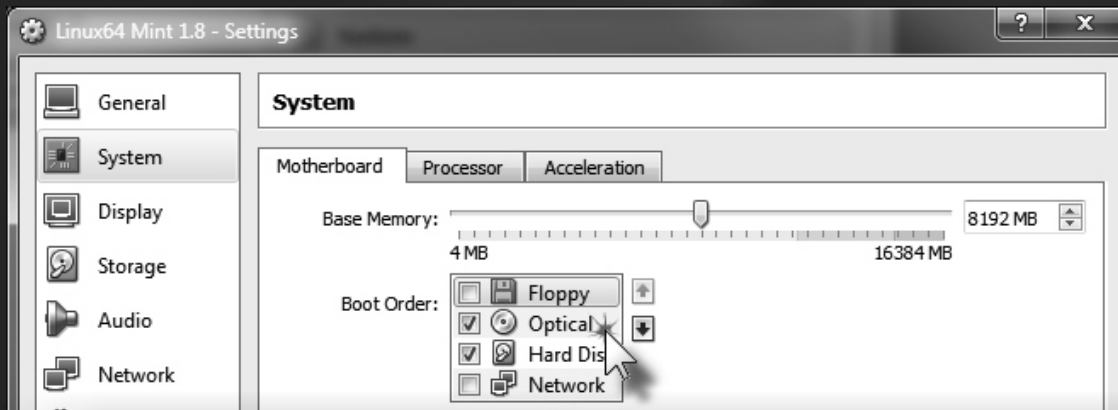
The size of the Hard disk is an open question. You best choose 40 GB because that is sufficient to really use the installation.

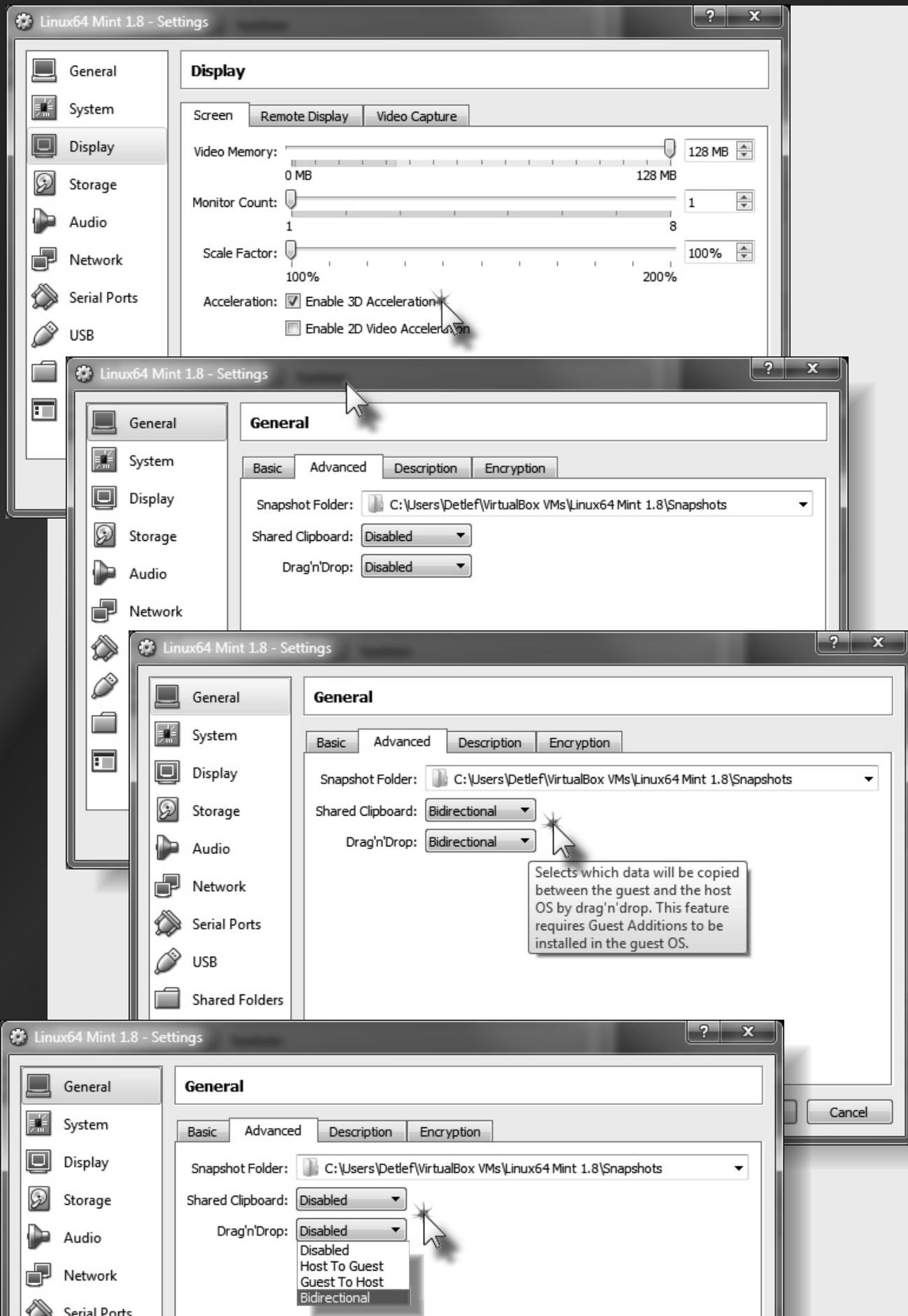
After these choices the Virtual Machine is created. We need to make additional changes to the standard options things are now organized in.

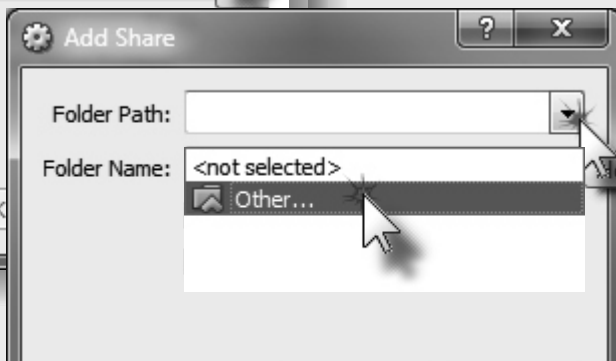
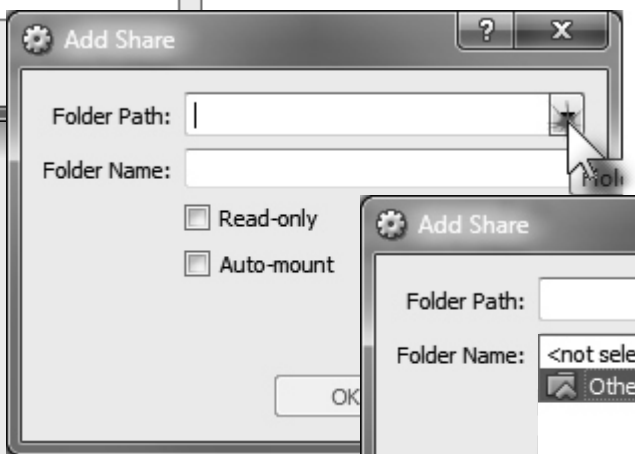
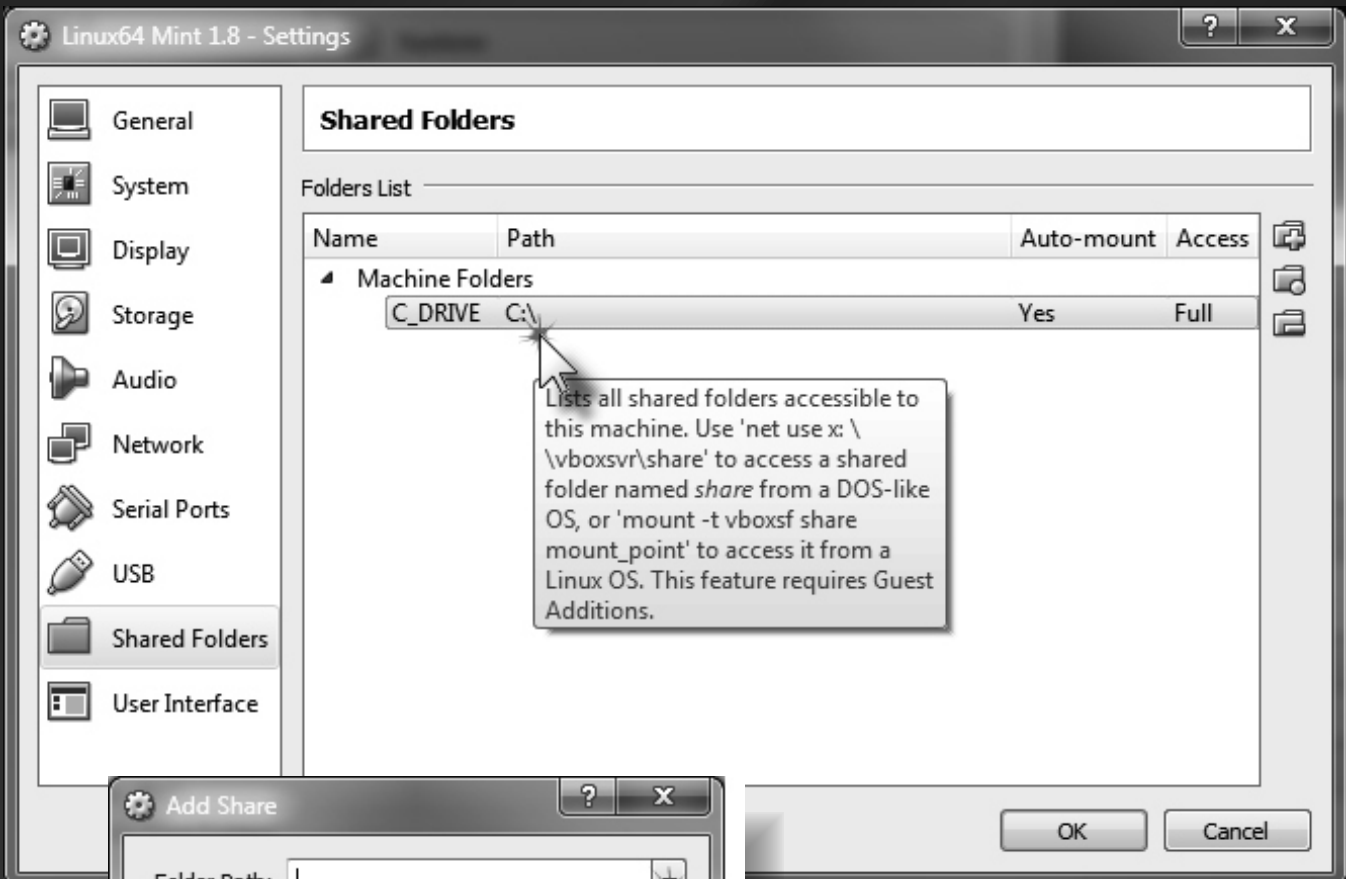


Just click on settings and the next window will appear...



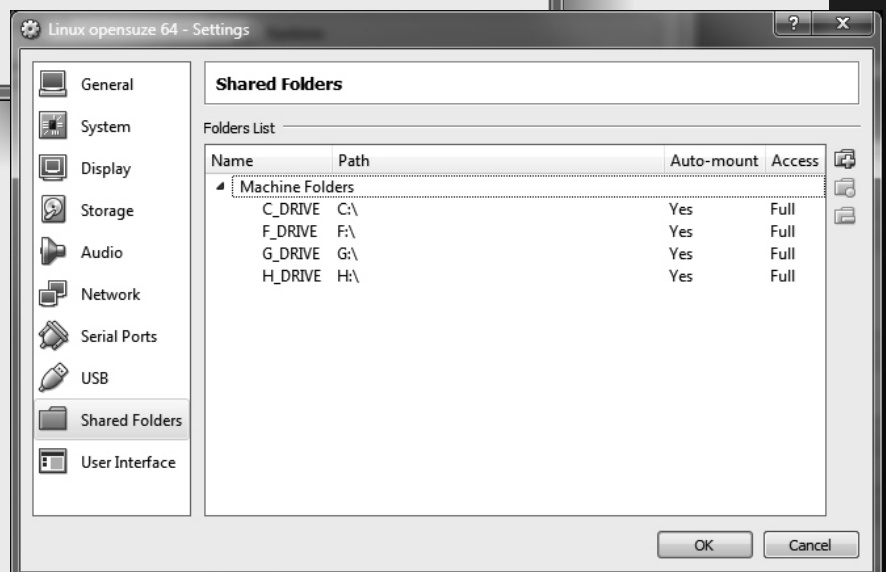


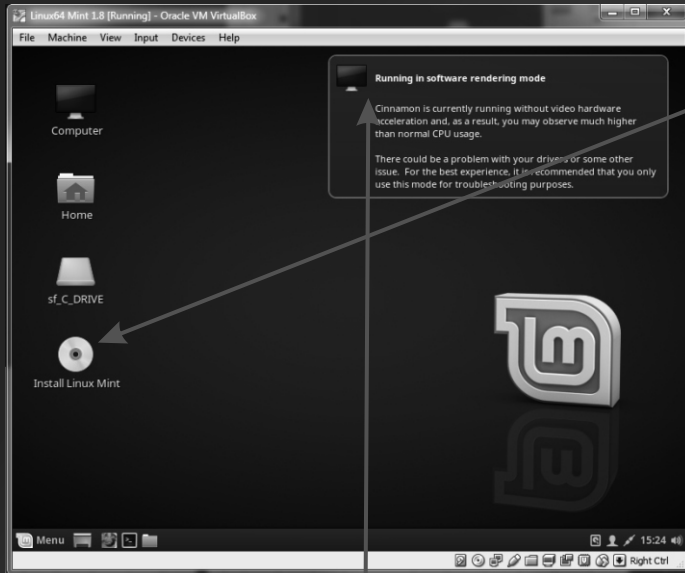




For the Folder path you need to click on Other. That will show a window where you can pick the path you would want to have available. You should realise that in case of connecting Linux and Windows the hard disk you have chosen will not become available. They have different file sorts...

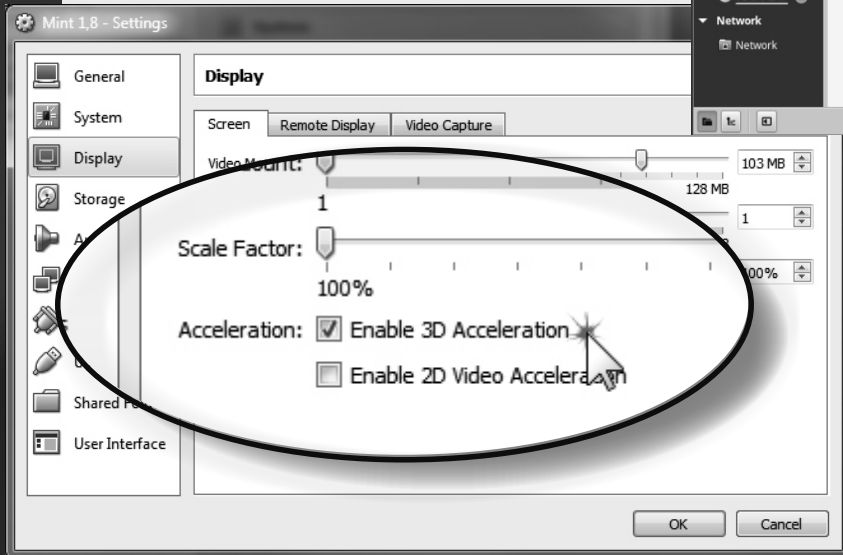
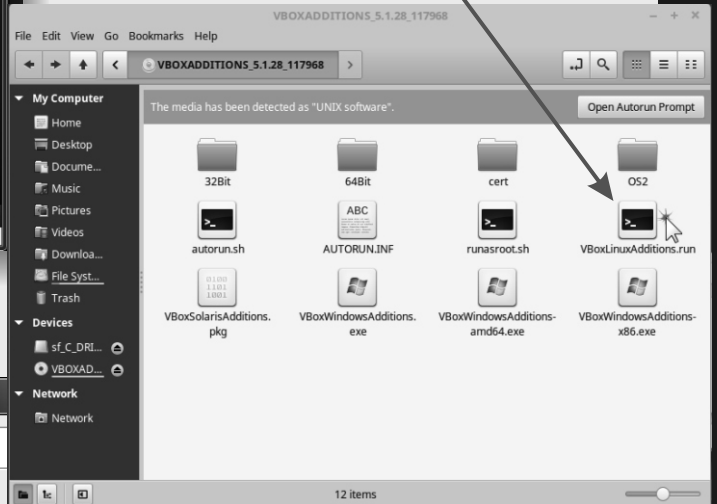
If you are working with Total Commander (File Commander help program) you could make a connection that will be shown.



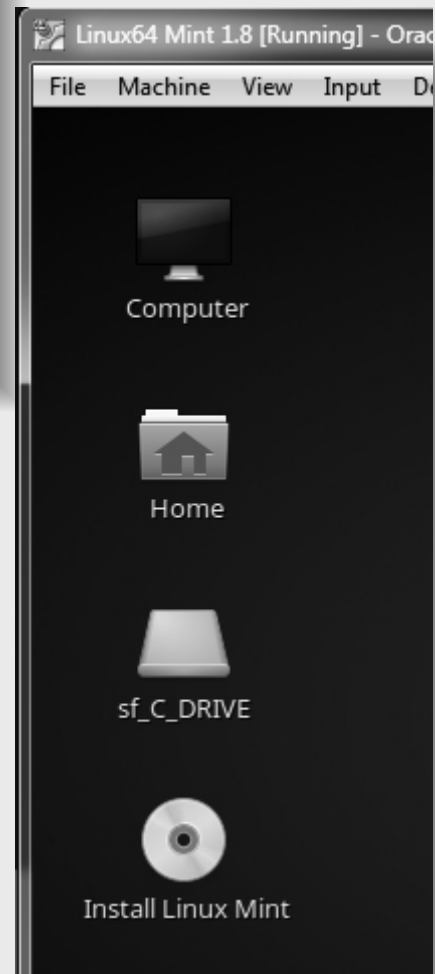
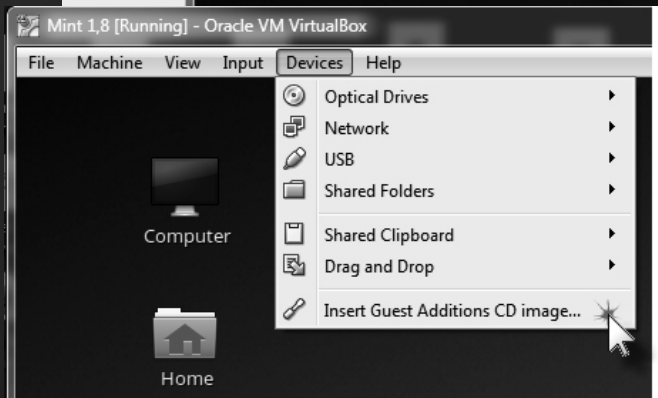
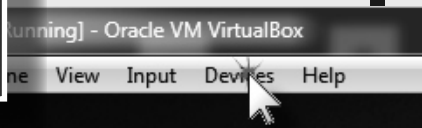
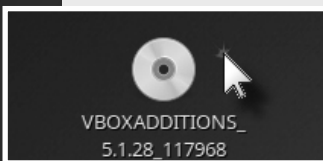


Sometimes the insertion fails. No problem. You could double click on the **Virtualbox Additions icon** at the left. In **Mint** they made it easy for you: the behavior is almost as in **Windows**. A new screen appears. It runs and then is automatically installed. Double click on the icon. You need to restart before it will work. The **Virtual Additions** have quite a lot of extra's that will work after installing it.

Because of my screen is 4 K it shows a note that it needs to install **VBOX ADDITIONS** and you also will have to activate the **3D Acceleration**.



Create the virtual box additions by looking for the additions to be available. Go to devices

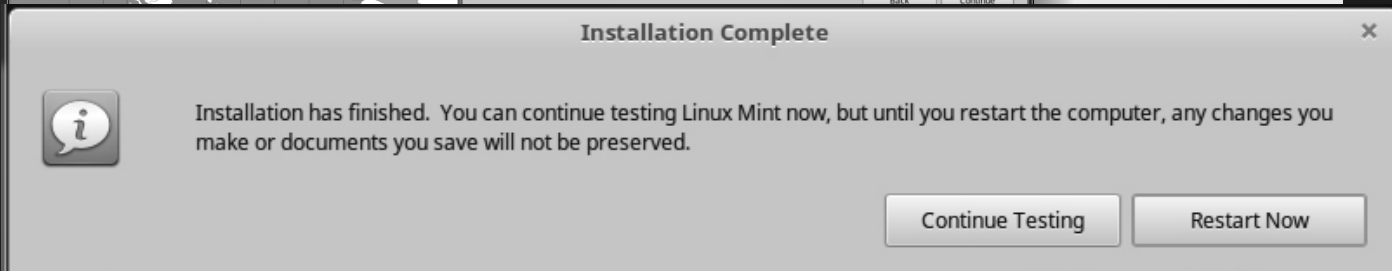
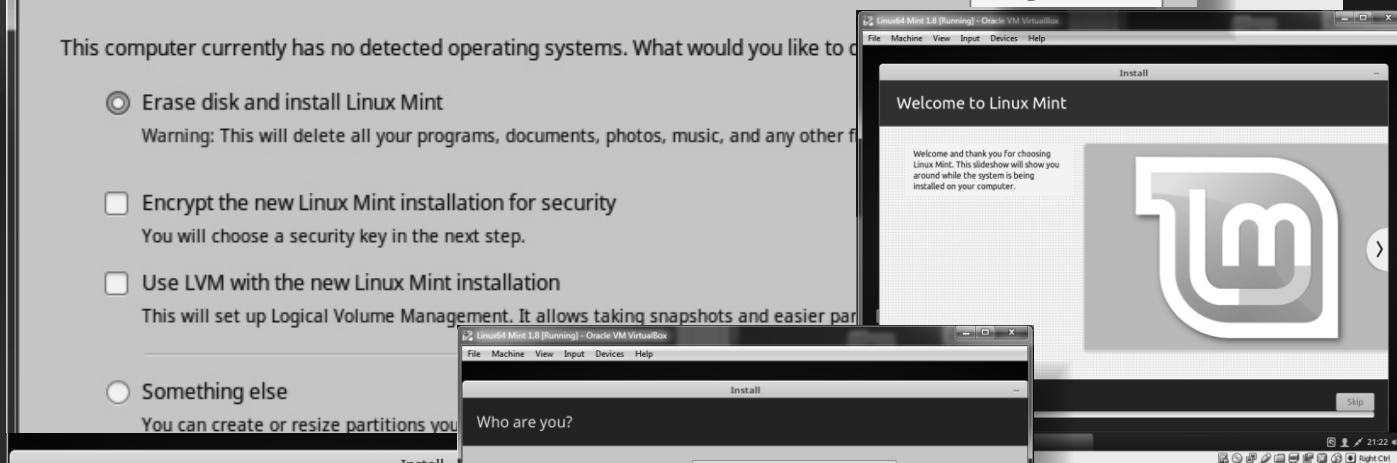




As soon you restart it will show the possibility to install Linux Mint



It will mention that it will install **Linux Mint** and warn you it will create the **virtual empty disk**. The Program will ask you the standard things where do yo live, what time table, what sort of keyboard. Enter your password name etc.





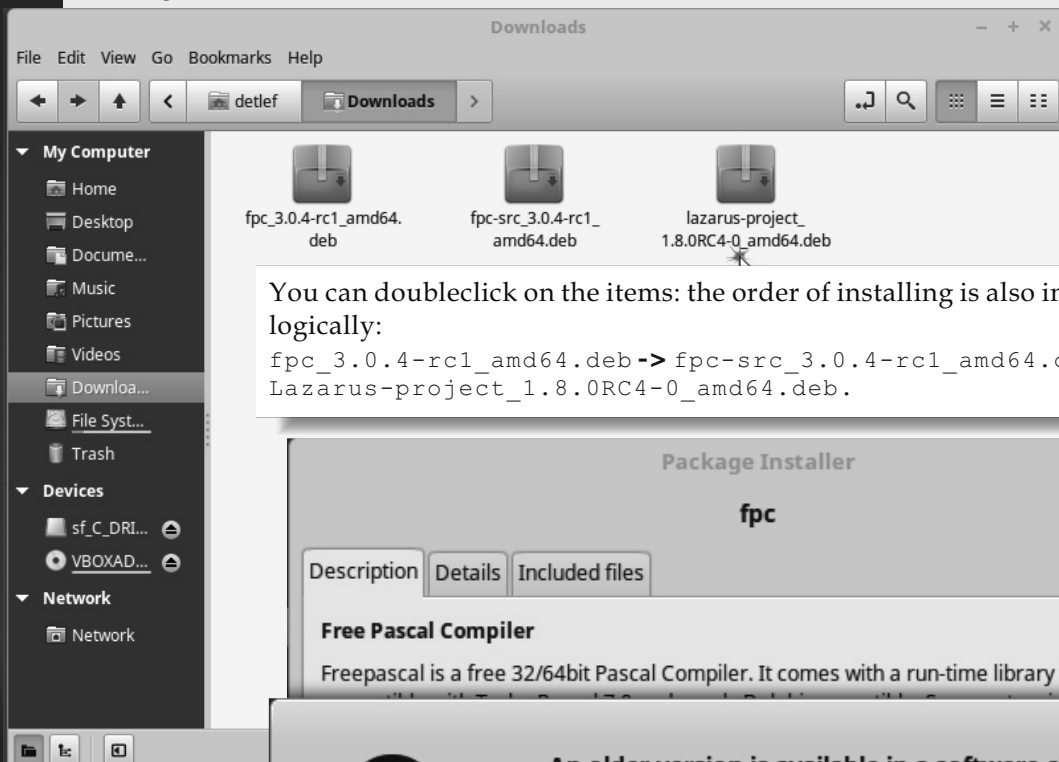
We almost have achieved our final goal: **Installing "Lazarus 1.8"**. The problem is that Mint has already a complete installation of an older version. So we need to follow this very carefully.... Double Click the icon "Home". A screen appears which contains a folder "Downloads" This folder will be empty. Now we can download the files we need (*we have them available in your own Blaise Pascal Magazine download section*). Just in case double click the **FireFox Web Browser Icon** , bottom left of the screen.

Go to the following address:

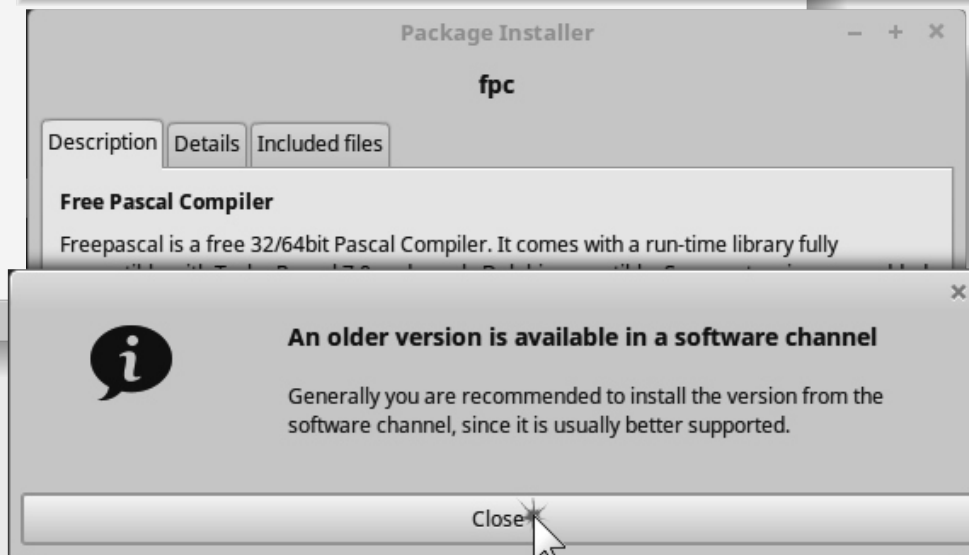
<https://sourceforge.net/projects/lazarus/files/Lazarus%20Linux%20amd64%20DEB/Lazarus%201.8.0RC4>

You need to download the 3 Files: `fpc_3.0.4-rc1_amd64.deb` / `fpc-src_3.0.4-rc1_amd64.deb` `lazarus-project_1.8.0RC4-0_amd64.deb` `fpc-src_3.0.4-rc1_amd64.deb`.

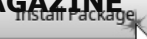
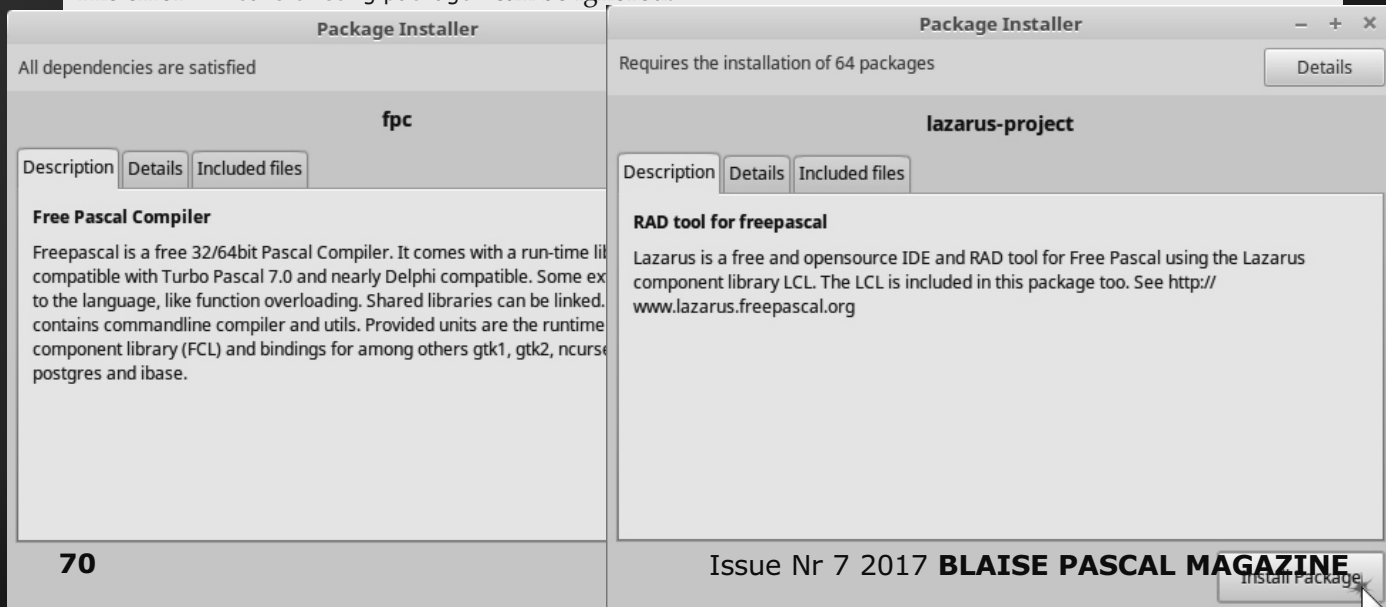
Take a good look at the files Free Pascal is RC 1 (release candidate 1) and Lazarus Rc4. That is correct.



You can doubleclick on the items: the order of installing is also important but logically:  
`fpc_3.0.4-rc1_amd64.deb -> fpc-src_3.0.4-rc1_amd64.deb` and the last `lazarus-project_1.8.0RC4-0_amd64.deb`.



Close the info and ignore that. Follow the instructions. Installing will follow. The error "Breaks existing package" can be ignored.





After all three files have been installed choose "Menu" at the bottom left and click on **programming**. The **Lazarus Icon** is available. Right click on the Icon and choose "Add to Desktop". Double click the **Lazarus icon** and the next screen becomes available: Everything seems to be installed correct, and now start Lazarus:

The image is a composite of two screenshots. The top screenshot shows the 'Configure Lazarus IDE' dialog box. The 'Lazarus' tab is selected, and the path '/usr/share/lazarus/1.8.0RC4/' is entered in the text field. The 'Start IDE' button is visible at the bottom right. The bottom screenshot shows the Lazarus IDE interface. The 'Source Editor' displays the following Pascal code:

```
unit Unit1;  
  Form1;  
  forms, Controls, Graphics, Dialogs;  
  
var  
  Form1: TForm1;  
20  
implementation  
  {$R *.lfm}  
25  
end.  
26
```

The 'Object Inspector' on the left shows the properties of 'Form1: TForm1', including 'Align' (alNone), 'AlphaBlend' (False), and 'Anchors' ([akTop,akLeft]).



starter

expert



## Abstract

A previous article provided an overview of the many dialogs available in the Lazarus Component Library (LCL), both component-based dialogs with an Execute function, and those called directly via a function or procedure interface (see Issue 64).

In spite of the rich variety of LCL dialogs available there are many situations where a customised dialog is the only solution for getting user input comfortably when multiple kinds of information are involved. Various aspects of providing this functionality are explored below, along with examples.

## Controlling numeric data input

Typically you encounter a situation where you need a mix of numeric and text data from the user. In theory you could gather it all as text via an uninspiring InputQuery() dialog to which you pass an array of prompts. But this dialog lets users type anything at all. You would then have to validate all the returned strings, handle error situations with invalid data and represent the dialog, possibly more than once. You also have to convert texts to numeric types. This is frustrating for both user and programmer. Far better to design a custom dialog that disallows invalid values at the point of entry, and returns data you know is immediately useable.

## A CROSSWORD EXAMPLE

Suppose you are developing a program that lets the user design and construct crosswords. At the outset you want to identify the crossword author, give the crossword a title, and specify its horizontal and vertical dimensions, so you know which words fit, and can eliminate from the dictionary of possibilities words that are too long. For display you also want the user to choose a suitable cell size in pixels (*a partially sighted user will need a bigger cell size and font, for instance*).

You need these five data items:

title, author (*of type String*);

and

cellSize, colCount, rowCount  
(*of type Integer*).

The obvious way to package this data is via a Pascal record. It might look like this:

```
TXWordData = record
  title: String;
  author: String;
  cellSize: Integer;
  colCount: Integer;
  rowCount: Integer;
  procedure Init(aTitle, anAuthor: String;
    aCellSize, aColCount, aRowCount: Integer);
  procedure DisplayData(aMemo: TMemo);
end;
```

**Note** the use of the advanced record functionality providing two procedures, one named **Init()** for easy initialisation of the record, and the other named **DisplayData()** used during the unit testing phase to display the record contents in a memo.

We need to include the compiler directive **{ \$ModeSwitch advancedrecords }** to activate this record functionality.

## About the author Howard Page-Clark

is a hobby programmer who learned Pascal in the days of Borland. After a career which included some years as a science teacher he works in retirement as a volunteer at a day centre, a secondary school, a psychiatric hospital and a church.







**CALLING A CUSTOM DIALOG**

What should the function signature look like that we use to call the new dialog?  
 One good approach is to make the routine a suitably named boolean function with an out record parameter filled by user interaction with the dialog. Following the call, if the function is True the out data parameter is valid; but if the function is False, it means the dialog was cancelled, and the data record will simply be full of zero values. In the crossword example given above the dialog function call would be:

```
function GetXWordDataDlg(out XWordData: TXWordData): Boolean;
```

The function result is set by comparing the modal dialog's **ModalResult** value with **mrOK**. The modal dialog (*a form class*) is designed to include a **TXWordData** record property named **XWordData**. This read-only property (*initially empty*) is filled with values entered by the user's interaction with the dialog when the OK button is clicked. If the dialog is cancelled a default record is returned. If the user fills and then accepts the dialog, the dialog's **XWordData** property value will be assigned to the calling function's out parameter. The full code for the calling function is:

```
function GetXWordDataDlg(out XWordData: TXWordData): Boolean;
var
  dlg: TXWordDialog;
begin
  XWordData:=Default(TXWordData);
  dlg:=TXWordDialog.Create(nil);
  try
    Result:=dlg.ShowModal = mrOK;
    if Result then
      XWordData:=dlg.XWordData;
  finally
    dlg.Free;
  end;
end;
```

Note how the dialog is created with no owner, and its memory allocation is protected with a **try ... finally ... end** construct which makes sure it is freed as soon as its task is complete. If the OK button is clicked to accept the dialog entries then the dialog's **XWordData** property is assigned to the function's out parameter. This style of function can serve as a general boilerplate template for any custom dialog requirement. The function has invented the **TXWordDialog** class. Now we need to implement it.

**DESIGNING TXWORDDIALOG**

The dialog can be designed as you would any additional form, whether modal or non-modal. The Lazarus IDE menu option File → NewForm will generate a default form unit. From the newly generated unit we first delete the global `Form2: TForm2` variable Lazarus supplies. Most likely your Lazarus settings are such that generating a new form also puts a `Application.CreateForm(TForm2, Form2);` statement in the project's `main.lpr` file to create the new form. If so, this line in the `.lpr` needs to be deleted  
 - make sure you delete the right one! Or you can achieve the same end by opening the project's **Project Options dialog**, opening the Forms page and moving the correct form from the Auto-create forms list to the Available forms list. Now we merely need to add appropriate widgets, labels and so on to our form, rename it as `XWordDialog` (*its type is then automatically changed to TXWordDialog*), and add a `XWordData` property and an `OKButton.OnClick` handler that populates the property correctly. Below the form's class declaration we add the signature of the calling function, and put its implementation in the unit's implementation section. The dialog that collects these five pieces of information is shown in Figure 1.

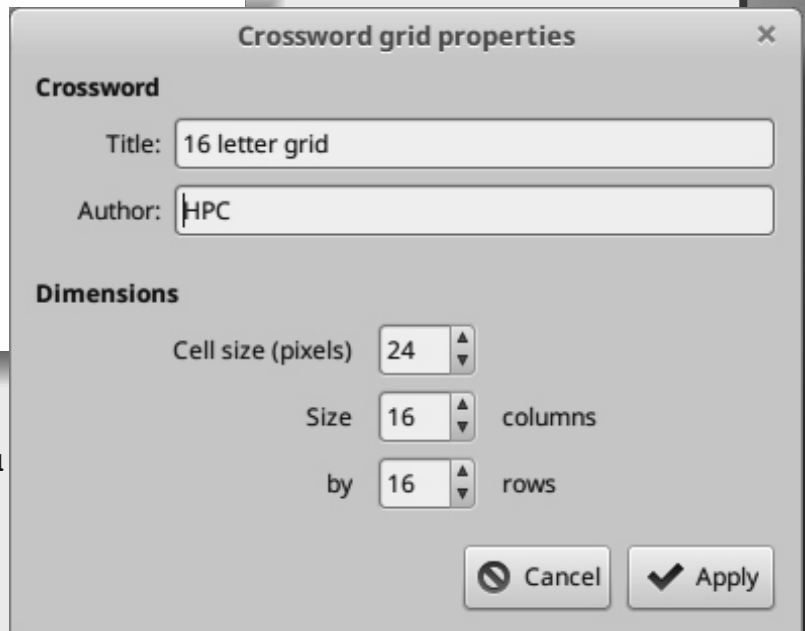
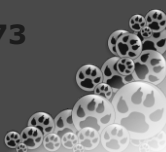


Figure 1 : The crossword dialog displaying





**NOTICE** the five edits for collecting the five pieces of information required, including three spinedits which give full control over the range and type of numeric information that can be entered.

The project that exercises the crossword dialog is called `Crossword.lpi`, and the dialog itself can be found in a unit called `uxwordddialog.pp` in the downloadable code. The project reports the values from the dialog as follows:

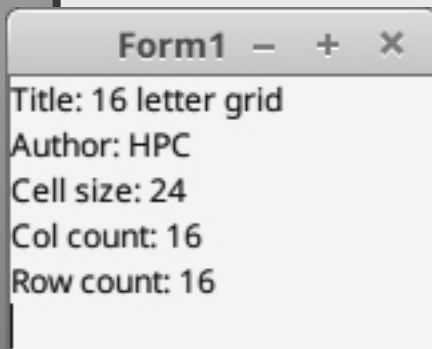


Figure 2 : Reporting dialog values

**A CUSTOM SETTINGS DIALOG**

A common situation where the need for a custom dialog arises is when obtaining, editing and storing a program’s settings and user-preferences.

You want to be able to present the user with a dialog that gathers all options within a single modal window where all the important settings can be selected with minimal need for typing or searching for particular options. The user can simply click labelled controls (*such as radiobuttons, checkboxes, sliders ...*) which are named descriptively to identify their functionality.

**DESIGNING THE UI**

If your program is of moderate complexity, you might choose a tabbed control as the main **GUI** container for the settings dialog, so that related settings can easily be grouped together on separate pages accessed by clicking the appropriate tab. Figure 3 shows a typical tabbed-dialog preferences GUI.

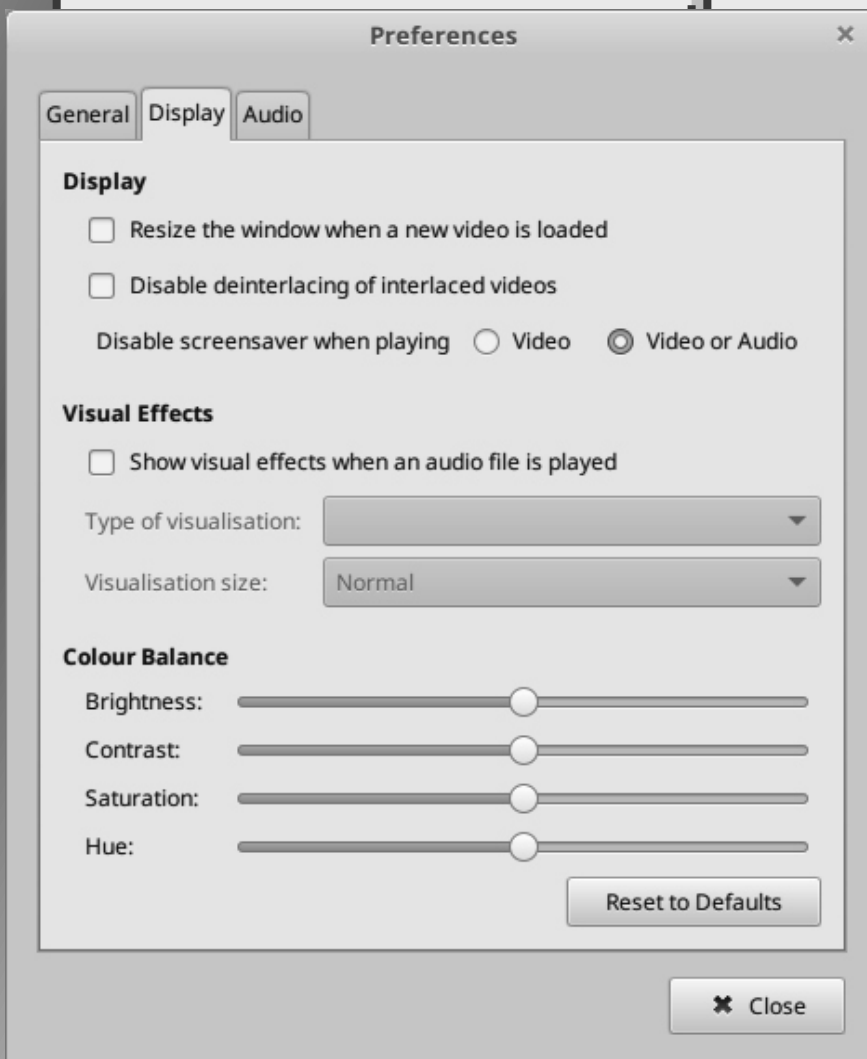


Figure 3 : A tabbed settings dialog from a multimedia app

The illustrated page makes use of:

- checkboxes
- radiobuttons
- comboboxes
- sliders
- standard buttons

to gather the required information from the user. Notice the single Close button at the bottom of the dialog. This follows a contemporary trend in UI design to minimise the number of buttons shown.

Here **Close** accepts the entered settings. The only way to the user can change her mind and cancel her choices is to click the system **close X icon** in the top right of the dialog, or perhaps press the **[Esc]** key.





To my mind this is a retrograde step in UI design, however popular it has become. The older convention of providing more than one named button, each clearly specifying its function seems a far clearer and more explicit approach, because the action of closing the dialog is the least important dialog action to describe. All users realise this is a **modal dialog** which will be closed when some button at the bottom is clicked (*other than Help if present, which of course should not close the dialog*).

The important question, not made explicit in this UI design is: "Will my chosen settings be saved or discarded?" A question that is not answered explicitly by this dialog design. The user clicks **Close** and hopes that her settings have been saved. But a lingering doubt remains, which could so easily be dispelled by naming the button Save settings (*or some such*) rather than Close. When the dialog disappears the user knows it has closed. But she still is not sure if her settings

were also saved... and why not provide a button labelled Discard changes?

**Not all innovations in UI design are improvements.**

To my mind explicit user feedback is far better than relying on an unwritten convention that a Close button also saves the data of a **modal dialog**.

A popular alternative to a tabbed control is to have a sidebar selection control (*often a treeview with nodes and sub-nodes*) for navigating the various settings pages. Where there are no sub-nodes to navigate, the sidebar may more simply be a panel or bar of large buttons. You might prefer a top bar with buttons arranged horizontally (*as are most tabs in tabbed controls*). Figure 4 illustrates a typical dialog of this sort where a user can tweak technical settings details. The page illustrated makes use of:

- comboboxes
- standard edit fields
- checkboxes
- standard buttons

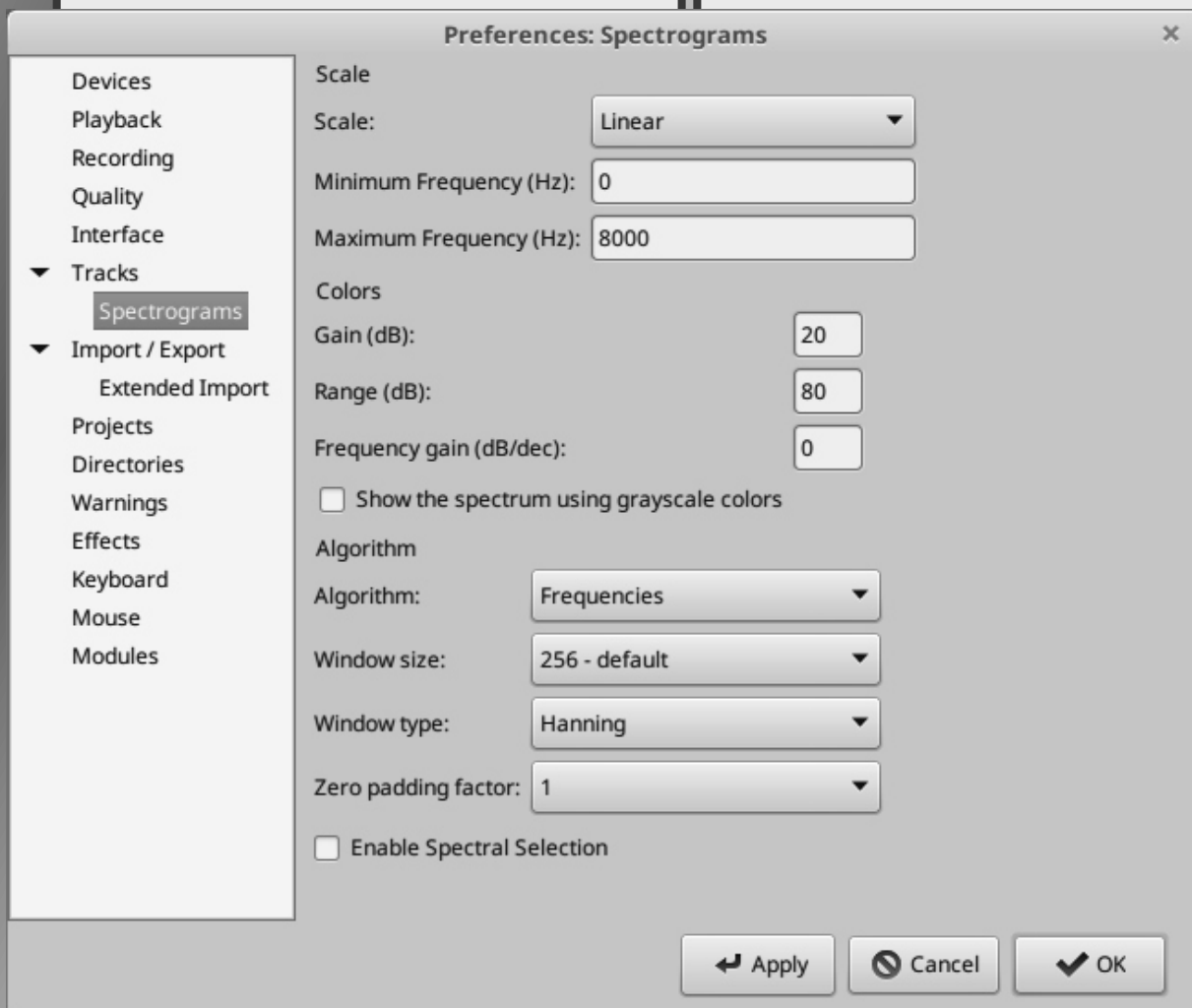


Figure 4 : Dialog with treeview sidebar for page navigation





**GUI WEAKNESSES**

Wherever possible you want the user to select from a list of possibilities rather than providing an edit field for them to type in.

For instance, where a default font needs to be set, you definitely don't want the user to type the name of the desired font. Far better to let the user choose from a list of available fonts than risk getting a wrongly spelled or non-existent font name returned.

But neither do you want a separate `TFontDialog` in addition to your main settings dialog. The font selection functionality is better integrated into the main dialog.

However, programmers often use the off-the-shelf solution of a `TFontDialog` component, making font selection and editing happen in a further modal dialog. Stacking modal dialogs on top of one another in this fashion is certainly possible, but can be a sign of lack of care in UI design.

**Note** in the illustrated dialog that numeric entries are obtained via standard edit controls. This is a **weakness in the design**, and forces introduction of **error dialogs** and **error procedures** where none would be needed if say a `spinedit` control had been used.

`Spinedits` exclude non-numeric entries by design, and the programmer also has complete control over the range of permitted values. Perhaps this looser editing of frequency, gain etc. shown in Figure 4 was adopted since it was assumed that only the most geeky users would ever tweak these settings, and they would know what they were doing...but even geeks make typos.

Also note how weak the separation of sections appears in Figure 4 (*they are named Scale, Colors, Algorithm*) since the section names are not in bold, or otherwise made to stand out. Compare this with the cleaner, more visually appealing section layout in Figure 3 where Display, Visual Effect and Colour Balance separate the sections well, without need of any divider line.

Also notice the small touch that Colour Balance appears in localised UK spelling in Figure 3, whereas Colors in Figure 4 has not been localised, displaying only the spelling familiar to the American programmer (*who perhaps designed the dialog*) as encountered in the USA.

Such details indicate some care has been taken over the design – the dialog in Figure 3 was not just thrown together, and user feedback has perhaps been taken on board to enhance the design.

Programmers know that captions have been included as resource strings to aid localisation for the end user, and that someone has taken the trouble to create and add translation files to the project.

**CUSTOM DIALOGS OF SOME COMPLEXITY**

*How do we go about implementing such complex dialogs in Lazarus?*

At least five questions need to be addressed at the start of the design:

1. What data is the dialog being designed to gather?
2. What will the calling interface look like? We answer this question by considering how the data the dialog gathers is passed back to the calling program.
3. Will the dialog need icons or other images, or will it be solely text-based? If images are needed, don't forget that designing icons or gathering the required images will be a distinct additional task.
4. Will the dialog be a persistent `lfm`-based window, or will it be a resourceless window created on-the-fly and destroyed as soon as it has been used? We answer this question partly by considering how difficult it is to design a good GUI without the aid of a RAD designer such as Lazarus provides, where you get immediate visual feedback about the layout and placement of control elements and how they fit into the overall design. You also need to gauge how often the dialog will be called during your program's lifetime. Is it likely not to be called, or to be called only once? Or is it quite likely that it will be reused several times? In the first case it makes more sense to create the dialog on-the-fly if it is ever required, and then release allocated memory and resources for the rest of the program's duration, so reducing your program's overall memory footprint. In the second case, you might consider it OK to have the dialog loaded at start-up and using allocated memory throughout the life of your program so it is instantly available when needed. If you know your program is to be used on older, low-spec machines the question of minimising resource usage may be critical if you need several complex dialogs and your program also manipulates large amounts of data.
5. How will the dialog be laid out: very simply, with multiple pages ...or in some other style? What Close-type buttons are required? Is a Help button needed? Are system icons wanted in the top window border?





**GOOD UI DESIGN IN CUSTOM DIALOGS**

As an example of good design layout consider the **Lazarus Anchors dialog**. It is **non-modal**, but illustrates good design for both modal and non-modal dialogs.

**AN LFM-BASED CUSTOM DIALOG**

Basing your custom dialog on a RAD form using the Lazarus form designer is the least code-intensive option for developing a custom dialog, and gives you maximum ability to tweak

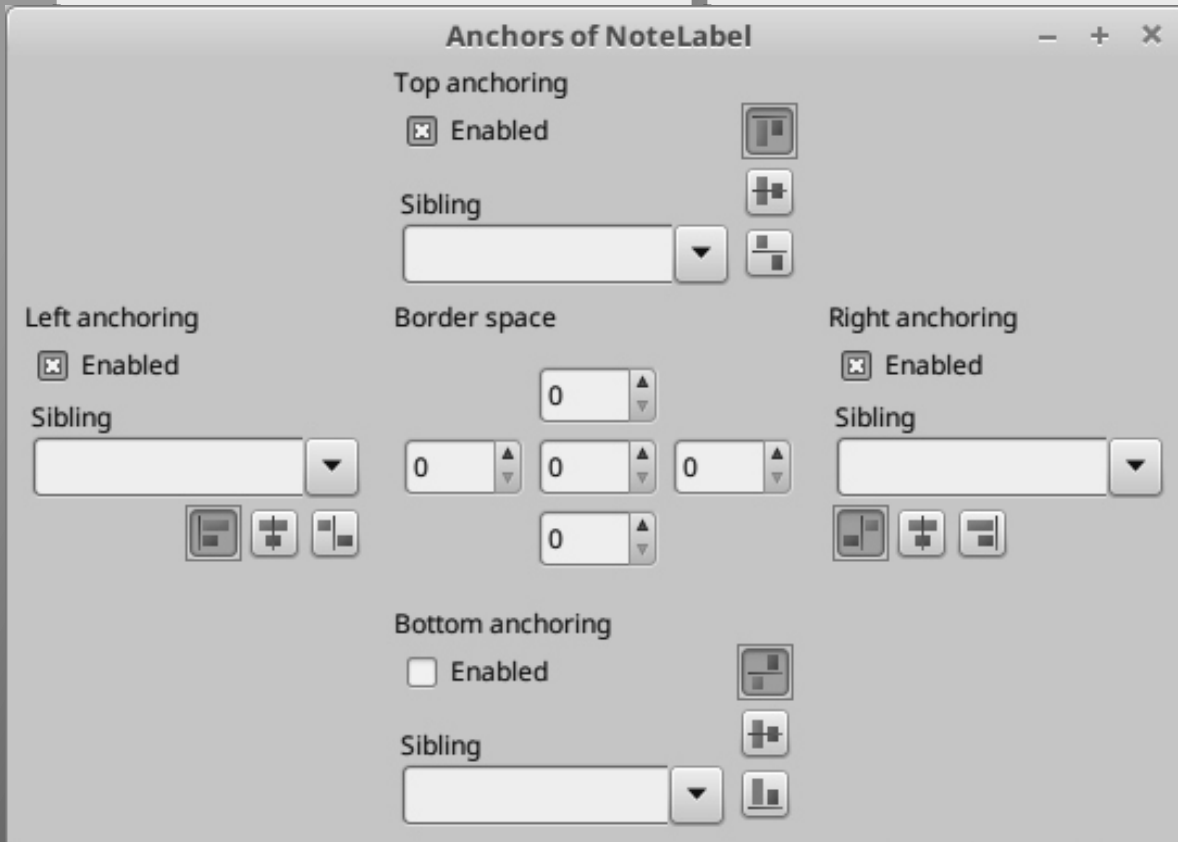


Figure 5 : The Lazarus Anchors dialog

The clean lines and logical placing of the control groups left, top, right, and bottom corresponding to the anchor being edited is obvious, a pattern repeated in the centre for the border-spacing spinedits.

Well-designed speedbutton icons obviate the need for descriptive button text that would clutter the interface. Integration with the IDE's object inspector is also tight and intuitive (*though the illustration cannot show that of course*).

An obvious enhancement would be for the groupbox titles (Left anchoring, Top anchoring etc.) to be in bold. This is not done only because of the restrictions of the native groupbox widgets used by Lazarus in the LCL. It is not possible to set the groupbox title font style separately from the font style of contained controls, whose captions (*Sibling, Enabled*) should not be emboldened. So you either write a custom groupbox control that does what you want, or learn to live within (*or work around*) the limitations of what native widgets offer.

layouts and styles, adjust critical properties and experiment with alternative controls while having continuous visual feedback which is nearly always **WYSIWYG**.

If we take Figure 3 as an example to copy, you would start with a new Lazarus project which you name **TestPreferencesDlg** and save the main form unit as **MainTestDialog**. Then via **FILE**→**New Form** generate a second form unit named **uPreferencesDlg**. Using the project's **Options dialog**, remove this as an auto-created form, and set the form's properties as follows:

Form property	Value
Autosize	True
Height	530
BorderIcons	[]
BorderStyle	bsDialog
Caption	Preferences
Name	PreferencesDialog
Position	poScreenCenter
Width	450





Drop a TPageControl on the form and set its properties as follows:

Form Property	Value
Align	alTop
BorderSpacing.Around	10
Name	PageControl
Height	450

Right-click on the page control and choose Add to add a General page named **tsGeneral**. Repeat this to add a Display page named **tsDisplay** and an Audio page named **tsAudio**. Select the **tabsheet** named **tsDisplay** and drop a number of controls on it to emulate Figure 3.

If you don't want to do this yourself, you can look at the

**upreferencesdlg.pp**

and

**upreferencesdlg.lfm**

in the code examples available for download from the **Blaise Pascal website**.

To emulate the dialog I used a mix of **TLabel**, **TCheckBox**, **TRadioButton**, **TComboBox**, **TTrackBar**, and **TButton**. Remove any global form variable.

This gives us the GUI for the dialog.

As with the simpler dialog in Figure 1 the best way to get user-data out of the dialog is to create a record as a data container.

A record of this type is then added as a private field to the dialog class, and made public via a read-only property. When the dialog is closed dialog's property value of the same type is assigned to the corresponding parameter of the function that called the dialog. The dialog can then be freed.

In this instance the Display page is collecting 10 separate data items. We can wrap the data items we want into a record called

**TDisplaySettings** that looks like this:

```
TDisplaySettings = record
  newVideoResize: Boolean;
  disableInterlacing: Boolean;
  disableScreenSaverBoth: Boolean;
  showVisualEffects: Boolean;
  visualisationKind: TVisualisationKind;
  visualisationSize: TVisualisationSize;
  brightness: Integer;
  contrast: Integer;
  saturation: Integer;
  hue: Integer;
  procedure Init(aNewVideoResize, aDisableInterlacing,
    aDisableScreenSaverBoth, aShowVisualEffects: Boolean;
    aVisualisationKind: TVisualisationKind;
    aVisualisationSize: TVisualisationSize;
    aBrightness, aContrast, aSaturation, aHue: Integer);
  procedure WriteToMemo(aMemo: TMemo);
end;
```

**TVisualisationKind** and **TVisualisationSize** are simple enums (see the source for dummy definitions).

You need to add the compiler directive **{ \$ModeSwitch advancedRecords }** to the **upreferencesdlg.pp** to enable the inclusion of procedures in the record.

The **Init()** procedure is used to simplify initialisation with data, and **WriteToMemo()** is used to display the contents of the record in the program that exercises the dialog.

**Init()** looks complicated only because it deals with ten data parameters at once.

How should the dialog be called by the program using it?

Our simplest option is to design a boolean function with this signature

```
GetDisplaySettingsDlg(out: TDisplaySettings): Boolean;
```

which returns *False* (and an empty parameter) if the dialog is cancelled, and returns *True* (with a fully populated **TDisplaySettings** parameter) if the user has accepted the dialog preferences.

The calling program can then proceed, ignoring a cancelled dialog, or dealing with the changed settings data resulting from user interaction with the dialog.

The calling function's implementation will look like this:

```
function GetDisplaySettingsDlg(
  out aDisplaySettings: TDisplaySettings): Boolean;
var
  dlg: TPreferencesDialog;
begin
  aDisplaySettings:=Default(TDisplaySettings);
  dlg:=TPreferencesDialog.Create(nil);
  try
    Result:=(dlg.ShowModal = mrOK);
    if Result then
      aDisplaySettings:=dlg.DisplaySettings;
  finally
    dlg.Free;
    dlg:=nil;
  end;
end;
```

The out parameter is first initialised using the recently introduced **Default() compiler intrinsic**. The dialog is then created within a **try . . . finally** construct to make sure it is properly destroyed after use. The dialog is then shown via **ShowModal**, whose result is compared to **mrOK**.





This comparison becomes the calling function's boolean Result. If the Result is True then the relevant data from the dialog's **DisplaySettings** property is copied to the out parameter for later use. All that remains is to provide an OnClick handler for the dialog's Close button that copies the ten required data items from the various dialog controls to the dialog's DisplaySettings property, and to remove the global **TPreferencesDialog** variable that Lazarus added when the form unit was generated. See the downloadable code for the details of this largely boilerplate code.

The **TestPreferencesDialog** project exercises this example custom dialog.

**NOTE** that adopting this UI design paradigm of a single Close button doubling as a Close and an OK button, and not providing a Cancel button (*or its equivalent*) means you have to write an **OnKeyDown** handler to trap the **[Esc]** key, rather than using the built-in functionality of a Cancel bitbutton of Kind **tkCancel**; with its Cancel property set to True. You must then also remember to set the dialog form's **KeyPreview** property to True. Otherwise your modal dialog will not respond to the **[Esc]** key, which is a cardinal programming sin.

### RESOURCELESS CUSTOM DIALOGS

If you are after a resourceless dialog (*one that does not need an .lfm at all*) you will have to do some extra work to duplicate the outcome of what Lazarus's streaming system would have done with the form's **.lfm**, had there been one.

If (*as here*) you already have a completed form resource, you can simply adapt the lfm text, massaging it into a constructed-in-code version, much in the same way as Lazarus does when building any form from its **.lfm** resource when it is first loaded.

**If you don't have any form resource as a basis to work from, you will have to build the code in a completely non-visual way.**

In either case, you construct the form not via an **inherited Create()** call, but via an **inherited CreateNew()** call.

You then create each control in code, set its required layout and other properties and parent each newly constructed control to the new resourceless dialog form. Once the new form is fully constructed you call its **ShowModal** method and proceed as before.

For complex dialogs with many (*possibly interacting*) controls, building the GUI involves many lines of setup code.

It is best to refactor these into a **BuildGUI** procedure invoked from the new form's constructor to aid program organisation.

Our boolean calling function, **GetDisplaySettingsNoLFMDlg()** will be almost identical to the earlier dialog invocation function, except this time the function will create (*and subsequently free*) a different dialog class, one that sets itself up without loading classes and property values from an **lfm** file. Instead it creates all needed widgets and sets their properties appropriately at the moment of invocation.

Again the downloadable code includes sources for the dialog created in this way, and a small calling program to exercise and test its functionality. Since several groups of controls needed a container with an emboldened title, I chose to develop a simple container that offered this functionality which I could reuse to avoid creating and setting properties for a title label over and over again. Some people prefer to use a frame for this, but I chose a custom control since I think it is usually more lightweight and flexible than a frame.

### CROSS-PLATFORM CONSIDERATIONS

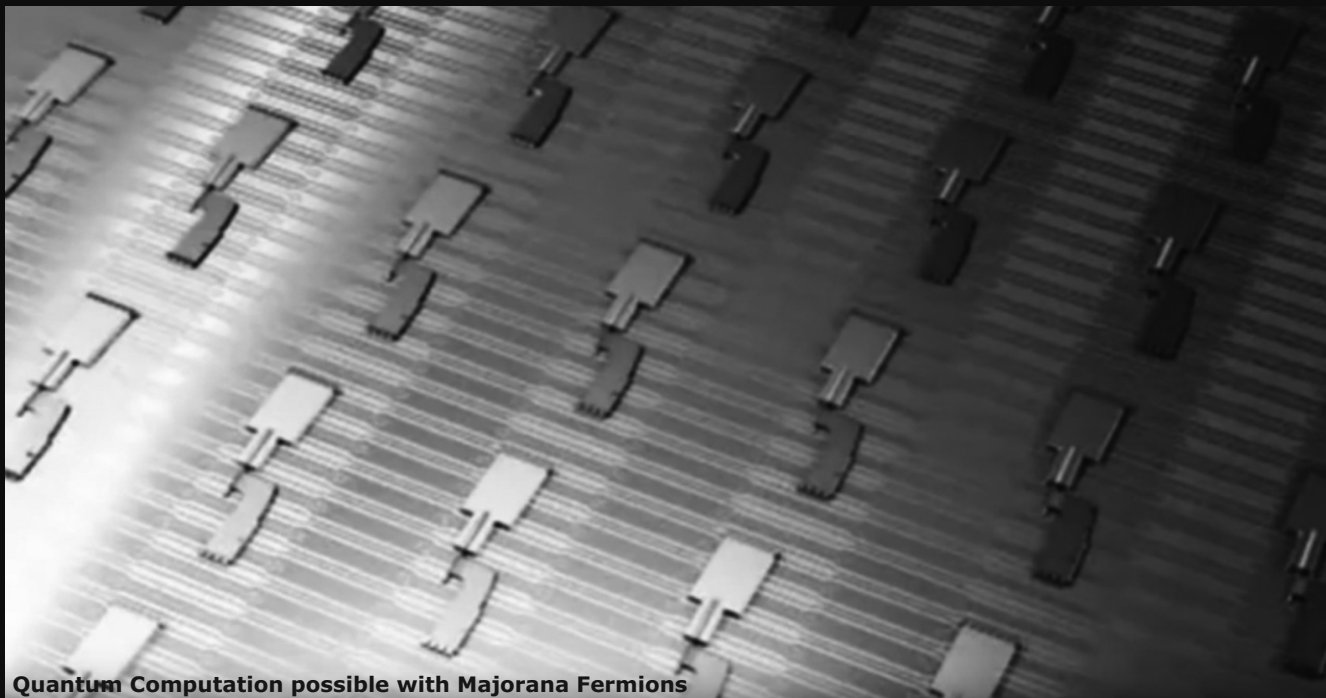
Resourceless dialogs can run aground on cross-platform issues to do with sizing and layout, particularly if you also cater for scaling at differing DPI settings. For instance most Linux slider widgets are slimmer than recent Windows equivalents. The chunkier Windows widgets can throw a careful layout off if it is designed first under Linux. **Usually use of anchoring combined with AutoSize overcomes these platform differences, but not always.**

The Lazarus autosizing algorithms are good but not perfect, and in some complex layout situations it can be tricky to get identical layouts on disparate platforms without hackish **{IFDEF xxx}** conditionals. If you are designing only for one platform you will probably avoid headaches that others have to address.

### EXAMPLE SOURCE CODE

The **Blaise Pascal website** has code for the examples above: the simpler **Crossword dialog**, and the more **complex Preferences dialog** in two versions – an **lfm-based** version and a **resourceless** version. The code compiles with the 1.6.4 release of Lazarus (*or any later version*). I apologise to readers of the previous article which should have made clear that the source code for the **Lazarus dialog exerciser** program described in that article would only compile with a trunk version of Lazarus or one of the release candidate versions of Lazarus 1.8, since it exercised not only the classic dialogs but the recently added **TTaskDialog**. Perhaps by the time you read this the 1.8 version of Lazarus will have been released.





Quantum Computation possible with Majorana Fermions


## KBMMW PROFESSIONAL AND ENTERPRISE EDITION

**V. 5.03.00 RELEASED!** NEW! AUTOMATIC DATABASE STRUCTURE UPDATE  
NEW! GENERIC OBJECT ORIENTED CONFIGURATION FRAMEWORK

- **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 pronouncable 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

 **COMPONENTS DEVELOPERS 4**



EESB, SOA, MoM, EAI TOOLS FOR INTELLIGENT SOLUTIONS. kbmMW IS THE PREMIERE N-TIER PRODUCT FOR DELPHI / C++BUILDER BDS DEVELOPMENT FRAMEWORK FOR WIN 32 / 64, .NET AND LINUX WITH CLIENTS RESIDING ON WIN32 / 64, .NET, LINUX, UNIX MAINFRAMES, MINIS, EMBEDDED DEVICES, SMART PHONES AND TABLETS.