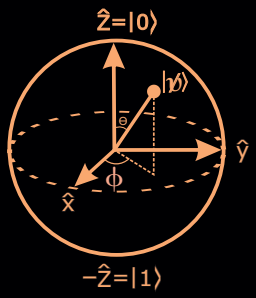
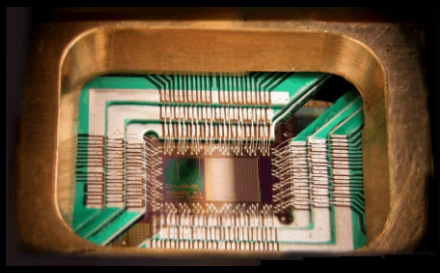
FOR DELPHI, LAZARUS, AND PASCAL RELATED LANGUAGES / ANDROID, IOS, MAC, WINDOWS & LINUX PRINTED, PDF, & ONLINE VIEW

BLAISE PASCAL MAGAZINE 62

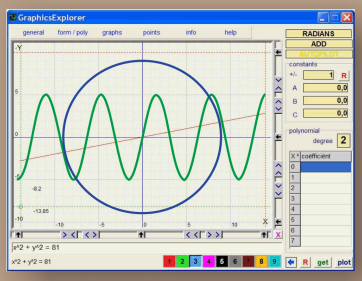


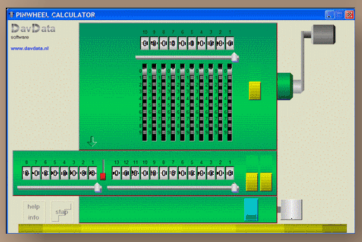


Quantum computing
By Editor
Books: Cross Platform Development
for Windows,Mac OS X (mac os) and LINUX
By Harry Stahl
Viruses without a trace
By Editor
Creating a ToDo list with kbmMW
By Detlef Overbeek
Direct Current (DC) networks project
a Delphi project to calculate currents and voltages in complex DC networks of resistors and voltages sources
By David Dirkse
Introduction to video processing
By Boian Mitov

PRINTED ISSUE PRICE €15,00
DOWNLOAD ISSUE PRICE € 5,00

DAVID DIRKSE

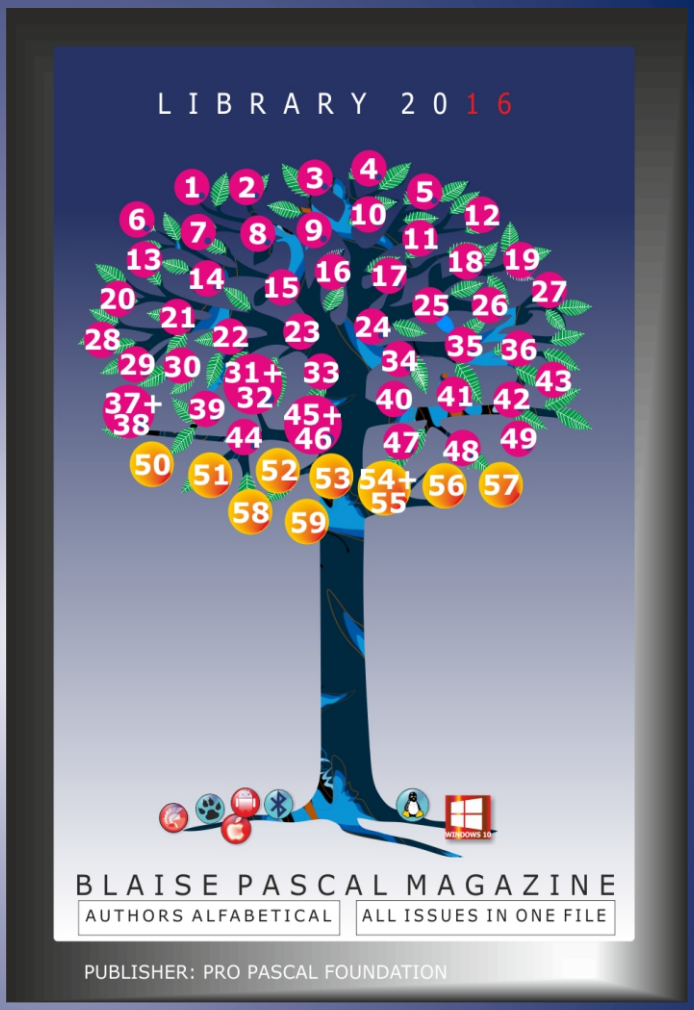




```
procedure ;
var
begin
 for i := 1 to 9
do
  begin
```

BLAISE PASCAL MAGAZINE
www.blaisepascal.eu

COMPUTER MATH & GAMES IN PASCAL



POCKET EDITION
Printed in full color.
A fully indexed PDF book is included.

THE FAMOUS LIBRARY STICK

GET THE BOOK INCLUDING THE NEWEST LIBRARY STICK INCLUDING 1YEAR DOWNLOAD FOR 75 EUROS + SHIPPING

NEWEST TECHNIQUE USB 3 / 16 GB. LIBRARY OF THE MAGAZINE UPDATED FROM ISSUE 1 TO ISSUE 59

COMBINATION: 3 FOR 1

BOOK INCLUDING THE LIBRARY STICK EXCL. SHIPPING INCLUDING 1YEAR DOWNLOAD FOR FREE

75€

http://www.blaisepascal.eu/subscribers/UK/UK_CD_DVD_USB_Department.html

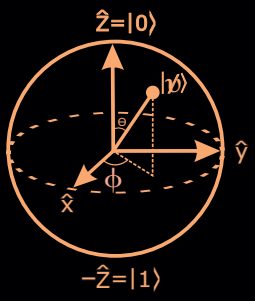
BLAISE PASCAL MAGAZINE 62

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

CONTENTS

ARTICLES

Quantum computing



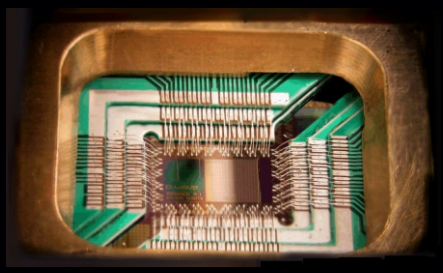


Image 1 Image 2

Image 1: The Bloch sphere is a representation of a qubit, the fundamental building block of quantum computers.
Image 2: Photograph of a chip constructed by D-Wave Systems Inc., mounted and wire-bonded in a sample holder. The D-Wave processor is designed to use 128 superconducting logic elements that exhibit controllable and tunable coupling to perform operations.

ADVERTISERS

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

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@blaisepascal.eu**

Editors
Peter Bijlsma, W. (Wim) van Ingen Schenau, Rik Smit,
Correctors
Howard Page-Clark, James D. Duff
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 (2013 prices)
1: Printed version: subscription € 85.-- Incl. VAT 6 % (including code, programs and printed magazine,
10 issues per year excluding postage).
2: Electronic - non printed subscription € 50.-- Incl. VAT 21% (including code, programs and download magazine)

Subscriptions can be taken out online at **www.blaisepascal.eu** or by written order, or by sending an email to **office@blaisepascal.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@blaisepascal.eu

From the Editor

Dear Reader,

This issue explores firstly the future of computing and secondly some present virus-related difficulties.

I will try to explain some of the alarming techniques that have been introduced in recent malware.

Following the article here about the quantum computing of the future, I plan to interview those researching quantum computing at the Technical University of Delft. In a subsequent issue I will report on how the model might work for an actual computing language, and what such a language might be.

A new quantum machine language?

A new Quantum Assembly?

Quantum Pascal?

How can we write real 'quantum' code? Although many people think it's going to be years before anything significant develops in the quantum computing field, I disagree. Having seen TU Delft's recent announcements and knowing the almost unlimited funding from their partners (Google) my question is not "How many years away?" but "How safe will it be?", given that it's probably coming soon.

And I think we should even now be considering the social and political impact of these likely developments. I will try to keep you informed about what is the present reality, as well as keeping you up to date with developments as they occur over the months ahead.

Espionage is around the corner and that makes our second topic of Viruses ever more relevant. MALWARE is actually a better term for this enemy of computing than virus, since virus does not do justice tot the threat this insidious software poses to all of us.

Some of these malware programs have such incredible names as "Posh Spy"! When I hear a name like that it takes me back to the science fiction of my youth: novels like "Viole Falushe", (written by Jack Vance - 1967) sounded much like this... very imaginative.

This malware is almost beautiful in its construction and very hard to discover, since it is incredibly well hidden.

I recently spoke with some of the people from Kaspersky Labs and so there will be a follow-up article about their experience of dealing with sophisticated new forms of malware.

There will be a solution to this problem, but as always it may take some time and be costly…

I hope you find the articles interesting.

We are laying plans for another **Pascon Delphi event on 19th September** this year at Eindhoven in the Netherlands, in the "Evoluon". This is the former science centre of "Philips" and has become world famous. For our Dutch and Belgian readers it is very easy to find. Eindhoven has its own airport which takes both domestic and international flights. We could not find a venue where your plane could land closer. Again we are booking expert speakers who will address the newest developments in Pascal Land. Forthcoming issues will whet your appetite with previews of what some of these speakers will showcase in detail at the Pascon event.

By the autumn our new website – https//www.blaisepascalmagazine.eu – will be operational and we will celebrate that with prizes you can win, and some special articles.

We are beginning work on some new books: a comprehensive manual (over 700 pages) for Lazarus and FPC, written by Michael van Canneyt and others aimed both at beginners and experienced users. We will keep you informed about progress.

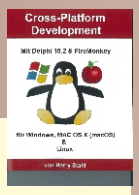
The Pascal2JS project is being tested and we will bring you further news about that as soon as possible.

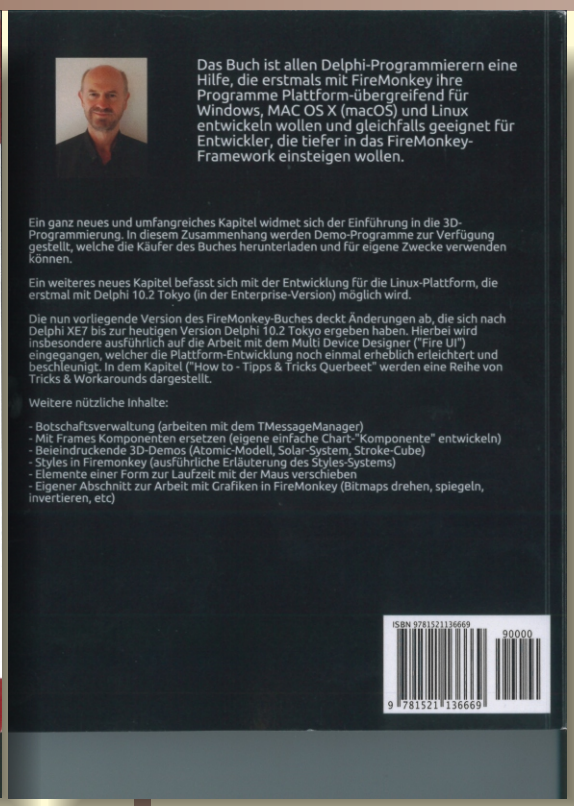
Howard Page-Clark is writing a Delphi book for complete beginners called Learn to program using Delphi.

If there are topics you would really like to read about, let us know.

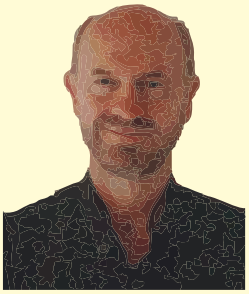
Maybe you noticed that each issue is now 10% bigger at 44 pages? Do let us know what you think of our plans and send us your feedback...

Detlef





Title of the book:
CROSS-PLATFORM DEVELOPMENT
MIT DELPHI 10.2 & FIREMONKEY FÜR WINDOWS, MAC OS X (MAC OS) & LINUX
This book is now available in German and soon will become available in English



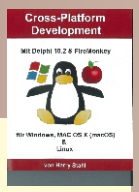
Author of the book:
Harry Stahl
Publisher : Harry Stahl
ISBN: 978-1521136669
Price : $ 50.96 / about € 40
Where to Buy: the best way to find the book is to go to Amazon and type in the ISBN number. It is found within a second.
https://www.amazon.com/Cross-Platform-Development-Delphi-FireMonkey-Windows/dp/1521136661

This book is aimed at Delphi programmers who want to use FireMonkey for the first time to turn their programs into Cross Platform Applications for Windows, MAC OS X (macOS) and Linux and of course for programmers that want to dive deeper into the FireMonkey-Framework.
A new and exhaustive Chapter will introduce you to the 3D programming aspects. In conjunction with this Demo programs can be downloaded by the book's purchaser so that he can use them for his own purposes.

A further new Chapter handles development for the **Linux-Platform,** which now for the first time is available in the Delphi 10.2 Tokyo Enterprise Version.

The currently available Version of the FireMonkey book also shows the changes made from **Delphi XE7** up to the current version of **Delphi 10.2 Tokyo**.
The **Multi Device Designer** ("*Fire UI*") is particularly well documented, which is a great help with development for non-Windows platforms and even accelerates coding. In the chapter "How to – Tips & Tricks Querbeet" a list of Tricks & Workarounds are shown.

FURTHER USEFUL CONTENTS:
- Message handling (*working with the TMessageManager*)
- Replace components with frames (*develop your own simple "Chart Components"*)
- Impressive 3D-Demos (*Atomic-Model, Solar-System, Stroke-Cube*) - Styles in FireMonkey (*thorough Explanation of the Style-System*)
- Moving elements of the form by the mouse during runtime.
- A special section about working with Graphics in FireMonkey (*Rotating, Mirroring, Inversioning etc of Bitmaps*)



CHAPTER 1 IN GERMAN:

Kapitel 1: Was ist FireMonkey

FireMonkey, üblicherweise mit "FMX" abgekürzt, ist eine Software-Komponenten -bibliothek bzw. ein Vektor basiertes Framework, womit Plattform unabhängige Anwendungen für Windows, MAC OS X (bzw. "macOS"), Linux, iOS und Android entwickelt werden können, oft mit dem gleichen Source-Code. Die erste FMX-Version wurde mit XE2 veröffentlicht, mit XE3 folgte eine stark erweiterte FMX-Version, die auch oft als FireMonkey 2 bezeichnet wurde.

Seitdem wurde FMX mit jeder Delphi-Version stark überarbeitet, so dass der Entwickler nicht selten eine Reihe von Anpassungen bei dem Umstieg auf die neueste FMX-Version machenmusste.

Erfreulicherweise nahm der Leistungsumfang mit jeder Version deutlich zu, so dass man heute ein sehr leistungsstarkes Framework hat, mit dem man nicht nur alles machen kann, was mit der VCL möglich ist, sondern darüber hinaus noch viel mehr.

Alle Komponenten sind frei rotierbar und einzeln skalierbar. Ferner existieren eine Reihe von 3D Komponenten, mit denen man 3D-Programme schreiben kann. Schließlich sind die Effekte und Animationen zu erwähnen, die FMX ein weiteres Alleinstellungsmerkmal verschaffen.

Die Darstellung der Komponenten wird i.d.R. von der GPU, also der Graphic Processing Unit, unterstützt, wodurch die Ausgabe schneller und flüssiger erfolgt. Unter Windows erfolgt die Ansprache der GPU mit DirectX, unter Mac mit OpenGL und unter iOS/Android mit OpenGL/ES.

Geschichte

FireMonkey wurde ursprünglich entwickelt von Eugene Kryukov (Firma KSDEV, Uland-UDE in Russland). Das Produkt war derzeit als VGScene bekannt. In 2011 wurde das Framework von Embarcadero aufgekauft und in Delphi ab der Version XE2 als neues Framework, neben der VCL, integriert.

Ab XE3 ist es lediglich ab der Enterprise-Version ein fester Bestandteil von Delphi, für die Professional -Version muss man es extra als sog. Mobile Pack erwerben.

Seit Delphi 10 Berlin kann man 64-Bit Anwendungen für Windows und auch für IOS erstellen, für MAC und Android bleibt es bislang bei der 32-Bit-Version. Ab Delphi 10.2 Tokyo wird auch die Linux-Plattform (64-Bit) unterstützt, allerdings nur zur Erstellung von Konsolen-Anwendungen.

Ausblick

Im Verhältnis zur VCL-Plattform finden die wesentlichen Neuerungen und Erweiterungen bei FMX statt. Es kommen immer wieder neue Komponenten und Eigenschaften zu den Komponenten hinzu. Insofern sehe ich hier die Zukunft der Software-Entwicklung mit Delphi.

Cross-Platform Development

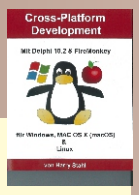
- 251 -

Stichwortverzeichnis

CONCLUSION:

At Amazon you can find the book and get a preview of the chapters by browsing through the chapters. First impressions of this book are very good and we will certainly use it to find out more about the quality. As soon the English version is available we will write more extensively about this excellent book.

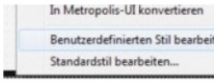
IT IS DEFINITELY WORTH ORDERING IT!



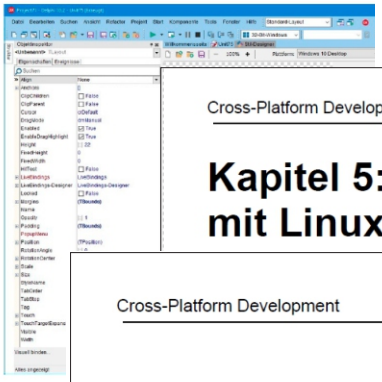
Abschnitt 5: Der FireMonkey Stil-Designer

a) Den Styles-Editor verwenden

Klicken Sie auf ein evtl. vorhandendes StyleBook dop
Maustaste auf ein Control und wählen den Befehl "Be
wenn Sie nur den Stil für die eine Komponente änder
bearbeiten", wenn Sie den Standard für alle Kompone
verändern möchten.



Sie erhalten dann folgende Ansicht:



Auf der linken
auswählen un
Strukturansicl
bzw. die Elem

Es geht aber
ausgewähltes

Abschnitt 6: MAC APIs (POSIX, CORE und Cocoa) in Delphi verwenden

Das MAC OS X Betriebssystem funktioniert im Wesentlichen mit 3 Layer-Systemen:

- POSIX
- CORE API
- COCOA Framework

Während die ersten beiden Layer über ein konventionelles C-Interface ansprechbar sind, ist das COCOA Layer über ein spezielles Objective-C Interface erreichbar. Eine Reihe von Funktionen steht letztlich in allen oder mehreren Schichten zur Verfügung. Zum Beispiel

abfragen oder
nderen

n an, die Sie

, mit der Posix-
s" definiert. Man
rt eben typische

lohnt sich, hier
Funktionen

Kapitel 5: Cross-Plattform Entwicklung mit Linux

Abschnitt 7: Für Linux nutzbare Delphi-Units

Hier erhalten Sie eine Übers
Platform verwenden können

System-Units

SysInit.pas
System.Bindings.Consts.pas: this
System.Bindings.CustomScope.pa
System.Bindings.CustomWrapper.
System.Bindings.EvalProtocol.pas
System.Bindings.EvalSys.pas
System.Bindings.Evaluator.pas
System.Bindings.Expression.pas
System.Bindings.ExpressionDefau
System.Bindings.Factories.pas
System.Bindings.Graph.pas
System.Bindings.Helper.pas
System.Bindings.Manager.pas
System.Bindings.ManagerDefaults
System.Bindings.Methods.pas
System.Bindings.NotifierContracts.
System.Bindings.NotifierDefaults.p
System.Bindings.ObjEval.pas
System.Bindings.Outputs.pas
System.Bindings.Search.pas
System.Bluetooth.Components.pas
System.Bluetooth.pas
System.Character.pas: this unit ha
System.Classes.pas: basic classes
System.ConvUtils.pas: convert uni
System.DateUtils.pas: dates proce
System.pas: the core unit
System.Diagnostics.pas
System.Generics.Collections.pas:
System.Generics.Defaults.pas
System.Hash.pas: hashing suppor
System.HelpIntfs.pas
System.IniFiles.pas: these clone th
System.Internal.DebugUtils.pas
System.Internal.ExcUtils.pas
System.Internal.JSONHlpr.pas
System.Internal.StrHlpr.pas
System.Internal.VarHlpr.pas
System.IOUtils.pas: support for mo
System.JSON.BSON.pas: this and
System.JSON.Builders.pas
System.JSON.Converters.pas
System.JSON.pas
System.JSON.Readers.pas
System.JSON.Serializers.pas

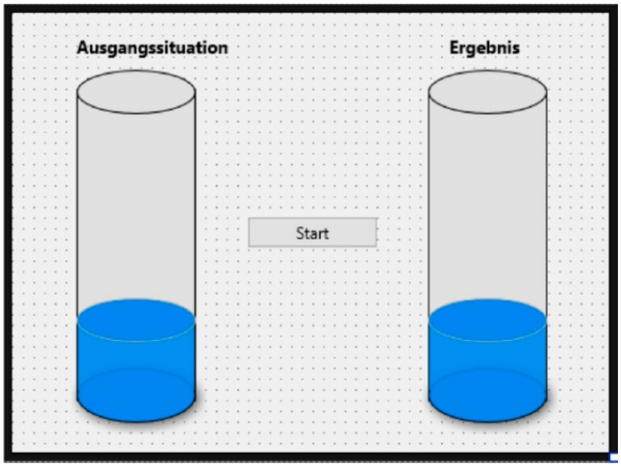
Kapitel 8: Animationen, Transitionen und Effekte

Bei verschiedenen Demos hier im Buch kamen schon FloatAnimationen zum Einsatz (Atomic Model und Solar Model).

Neben der FloatAnimation ist auch die TColorAnimation interessant. Damit können Sie eine Farbe in eine andere Farbe wechseln lassen in einer von Ihnen gewünschten Zeitdauer.

Hier mal ein kleines Demo, welches einen Laborversuch simulieren soll, wo in einem Reagenzglas eine blaue Flüssigkeit in eine gelbe Flüssigkeit übergeht.

Dieses Demo-Projekt verdeutlicht gleichzeitig, mit wie wenig Aufwand Sie interessante Ergebnisse erzielen können. So sieht das Projekt im Design-Modus aus:



In der Strukturübersicht können Sie erkennen, wie die Objekte konstruiert worden sind:

BY THE EDITOR

ABSTRACT

This article attempts to explain some of the most important elements of Quantum Computing, and if and when the first quantum computers are going to be ready for use. This article has been written with the help of Wiki and the help of the site Quantum made simple:
http://toutestquanique.fr/en/
This site has very attractive video-explanations of a great deal to do with Quantum Computing.
We would encourage you to visit it.

INTRODUCTION

We will explain some of the basic elements, such as needing to differentiate between Yes and No (or 1 and Ø at the binary level) in computing.
One of the more interesting questions is: does the boolean data type still work, and what is its result?

Since quantum bits (qubits) experience many stages through their lifetime, it might very well be that we would be able to add a totally new experience. Humans know the options of "yes" and "no" as well as "maybe."
Qubits have enough lifetime stages that it might very well be that in future we could have new states: "Maybe Not" and "Maybe Yes".

It sounds odd enough to be a revolution in computing, but this is an area which currently lacks sufficient philosophical research. So the world is not "black and white" after all, as we know of course.

Realising the goal of building a quantum computer will have very significant consequences. It is possible, because of the multiple manifestations of the qubit that we will be unable to create this very much wanted machine because the Qubit might be more complex and elusive than we yet have found. In that case it might be very difficult to achieve what we want.

(**QUBITS** *In quantum computing, a qubit or quantum bit (sometimes qbit) is a unit of quantum information – the quantum analogue of the classical bit. A qubit is a two-state quantum-mechanical system, such as the polarization of a single photon. Here the two states are vertical polarization and horizontal polarization. In a classical system, a bit would have to be in one state or the other. However, quantum mechanics allows the qubit to be in a superposition of both states at the same time, a property that is fundamental to quantum computing.)*

WHAT IS QUANTUM COMPUTING?

Quantum computing studies theoretical computation systems (quantum computers) that make direct use of quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. Quantum computers are different from binary digital electronic computers based on transistors. Whereas common digital computing requires that the data be encoded into binary digits (bits), each of which is always in one of two definite states (0 or 1), quantum computation uses quantum bits, which can be in superpositions of states. A quantum Turing machine is a theoretical model of such a computer, and is also known as the universal quantum computer.

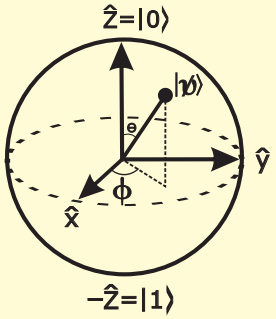


Figure 1:
The Bloch sphere is a representation of a qubit, the fundamental building block of quantum computers. See further explanations Figure 2.

(**COMPUTATION** *is any type of calculation that includes both arithmetical and non-arithmetical steps and follows a well-defined model understood and described as, for example, an algorithm. The study of computation is paramount to the discipline of computer science.)*

(**QUANTUM SUPERPOSITION** *is a fundamental principle of quantum mechanics. It states that, much like waves in classical physics, any two (or more) quantum states can be added together ("superposed") and the result will be another valid quantum state; and conversely, that every quantum state can be represented as a sum of two or more other distinct states. Mathematically, it refers to a property of solutions to the Schrödinger equation; since the Schrödinger equation is linear, any linear combination of solutions will also be a solution. An example of a physically observable manifestation of superposition is interference peaks from an electron wave in a double-slit experiment. Another example is a quantum logical qubit state, as used in quantum information processing, which is a linear superposition of the "basis states" $|0\rangle$ and $|1\rangle$. Here $|0\rangle$ is the Dirac notation for the quantum state that will always give the result 0 when converted to classical logic by a measurement. Likewise $|1\rangle$ is the state that will always convert to 1.)*

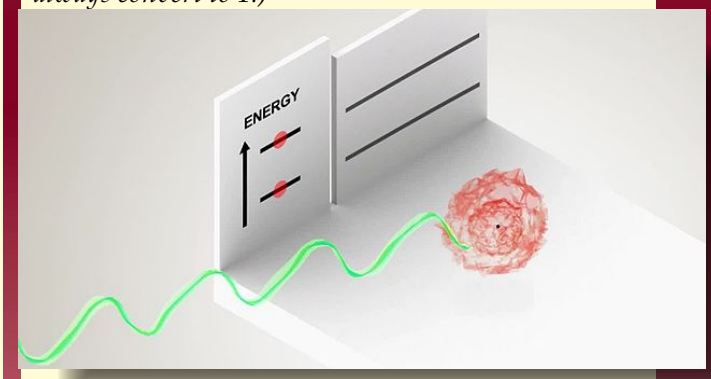


Figure 2. The green line shows the "wave" which means the superposition changing through time. The two red circles show the superpositions of the Qubit: the outer ring is active state, the inner is the inactive state. They can be active and inactive at the same time.

If you are interested you will find a video on this subject in the PDF file of the magazine. Simply double-click on the item. (Figure 2) Otherwise you can use this address to see that video through YouTube:
`https://youtu.be/7B11lCxVdkE`

Some quantum systems, such as atoms, photons, or spins, can be in two simultaneous different states. We call these **"SCHRÖDINGER'S CATS"**.

In the lab, we observe quantum superposition in a great number of systems: atoms, photons, spins, etc. Studying its appearance and its frailty is important to better understand the underlying principles of quantum mechanics. It can also operate as a detection device, de-coherence being used to detect other quantum objects such as a spin. Superposition is also the elementary unit, the "quantum bit", thanks to which we can perform quantum calculations, and it is at the core of quantum information and quantum computing research.

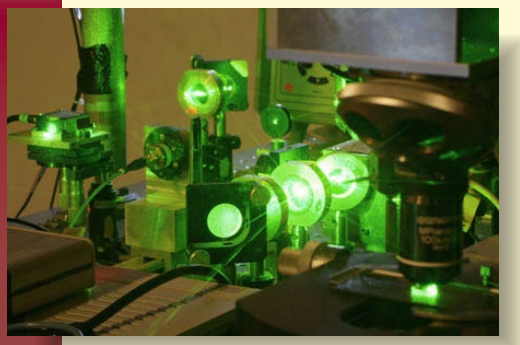


Figure 3: The lab showing some laser techniques to separate the state of the qubits

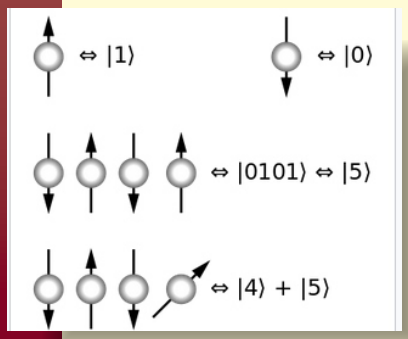


Figure 4: Qubits are made up of controlled particles and the means of control (e.g. devices that trap particles and switch them from one state to another).

A **"Quantum Turing machine"** is a theoretical model of such a computer, and is also known as the **UNIVERSAL QUANTUM COMPUTER.**

(**ENTANGLEMENT** *is a physical phenomenon that occurs when pairs or groups of particles are generated or interact in ways such that the quantum state of each particle cannot be described independently of the others, even when the particles are separated by a large distance – instead, a quantum state must be" described for the system as a whole.)*

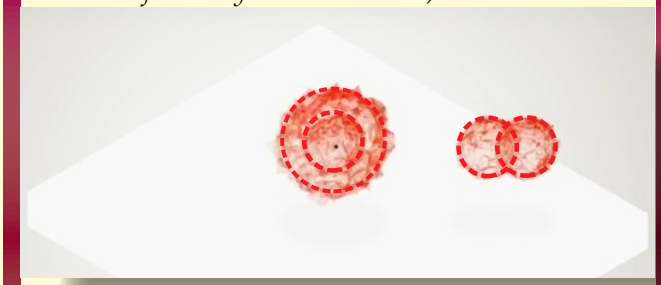


Figure 5: The Bloch sphere is a representation of a qubit, the fundamental building block of quantum computers.

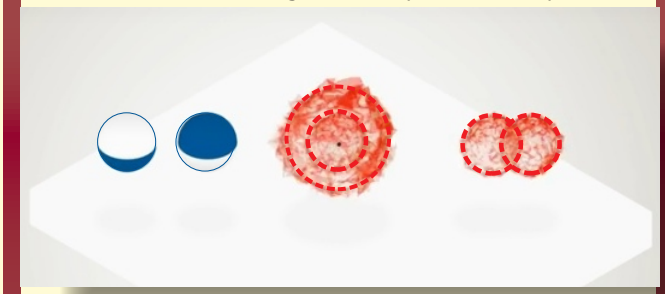


Figure 6: This ability to a superposition of state applies to any quantum particle. For example a molecule, a photon or a spin , (a small magnetic effect carried by electrons).

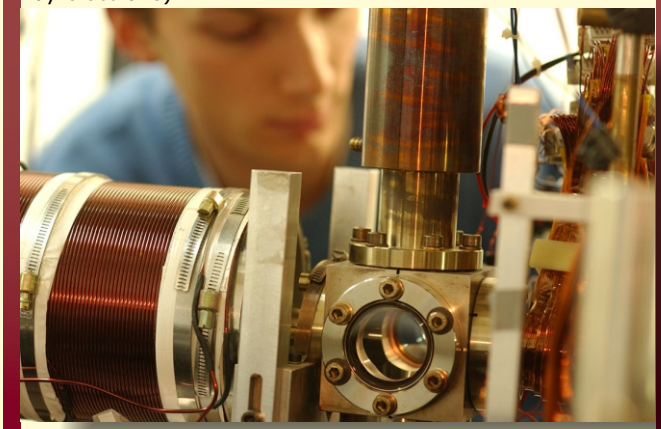


Figure 7: In the laboratory- Bose-Einstein condensates are created in quantum optics and atomic physics labs. Not only are they used to reproduce complex situations in solids, but also to study some fundamental quantum properties. They may even be used to manipulate, slow down or stop light. An equivalent of condensates, such as superfluid helium or certain magnetic materials when put in high magnetic fields can also be created in other systems

As of 2017, the development of actual quantum computers is still in its infancy, but experiments have been carried out in which quantum computational operations were executed on a very small number of quantum bits. Both practical and theoretical research continues, and many national governments and military agencies are funding quantum computing research in an effort to develop quantum computers for civilian, business, trade, environmental and national security purposes, such as cryptanalysis.

(CRYPTANALYSIS *(from the Greek kryptós, "hidden", and analýein, "to loosen" or "to untie") is the study of analysing information systems in order to study the hidden aspects of the systems. Cryptanalysis is used to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the cryptographic key is unknown. Matthijs Coster wil explain some of that in an upcoming article by him. He is studying Quantum cryptanalysis.)*

Large-scale quantum computers would theoretically be able to solve certain problems much more quickly than any classical computers that use even the best currently known algorithms, like integer factorization using **Shor's algorithm** or the simulation of quantum many-body systems.

There exist quantum algorithms, such as **Simon's algorithm**, that run faster than any possible probabilistic classical algorithm.
A classical computer could in principle (*with exponential resources*) simulate a quantum algorithm, as quantum computation does not violate the Church–Turing thesis. On the other hand, quantum computers may be able to efficiently solve problems which are not practically feasible on classical computers.

(QUANTUM SIMULATION - *The idea that quantum computers might be more powerful than classical computers originated in Richard Feynman's observation that classical computers seem to require exponential time to simulate many-particle quantum systems. Since then, the idea that quantum computers can simulate quantum physical processes exponentially faster than classical computers has been greatly fleshed out and elaborated.*

Efficient quantum algorithms have been developed for simulating both Bosonic and Fermionic systems, and in particular the simulation of chemical reactions beyond the capabilities of current classical supercomputers and this requires only a few hundred qubits.

Quantum computers can also efficiently simulate topological quantum field theories. In addition to its intrinsic interest, this result has led to efficient quantum algorithms for estimating quantum

In quantum mechanics, a boson is a particle that follows Bose–Einstein statistics. Bosons make up one of the two classes of particles, the other being fermions. The name boson was coined by Paul Dirac to commemorate the contribution of the Indian physicist Satyendra Nath Bose in developing, with Einstein, Bose–Einstein statistics—which theorizes the characteristics of elementary particles.

Figure:8 Satyendra Nath Bose

A classical computer has a memory made up of bits, where each bit is represented by either a one or a zero. A quantum computer maintains a sequence of qubits.
A single qubit can represent a one, a zero, or any quantum superposition of those two qubit states; a pair of qubits can be in any quantum superposition of 4 states and three qubits in any superposition of 8 states. In general, a quantum computer with n qubits can be in an arbitrary superposition of up to 2 n different states simultaneously (*this compares to a normal computer that can only be in one of these 2 n states at any one time*).
A quantum computer operates by setting the qubits in a perfect drift that represents the problem at hand and by manipulating those qubits with a fixed sequence of quantum logic gates . In quantum computing and specifically the quantum circuit model of computation, a quantum gate (*or quantum logic gate*) is a basic quantum circuit operating on a small number of qubits. They are the building blocks of quantum circuits, like classical logic gates are for conventional digital circuits.
The sequence of gates to be applied is called a quantum algorithm. The calculation ends with a measurement, collapsing the system of qubits into one of the 2 n pure states, where each qubit is zero or one, decomposing into a classical state. The outcome can therefore be at most n classical bits of information.

Quantum algorithms are often probabilistic, in that they provide the correct solution only with a certain known probability. Note that the term non-deterministic computing must not be used in that case to mean probabilistic (computing), because the term non-deterministic has a different meaning in computer science.
An example of an implementation of qubits of a quantum computer could start with the use of particles with two spin states: "down" and "up" (typically written $|\downarrow\rangle$ and $|\uparrow\rangle$ or $|0\rangle$ and $|1\rangle$). This is true because any such system can be mapped onto an effective spin-1/2 system.

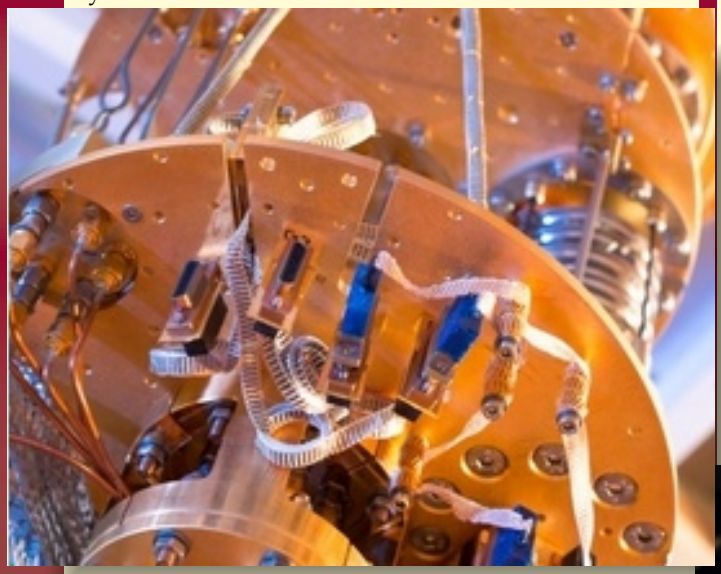


Figure 8

THE REALITY OF TODAY
The Technical University of Delft (TuDelft) is one of the leading University's on Quantum Technology.
It is a key future emerging technology. QuTech is at the forefront of research and development in quantum technology. QuTech currently has three research & technology roadmaps and one partnering roadmap. Additional roadmaps will be developed in the next five years. Google has become one of their partners.

Fifteen years ago, the quantum world was limited to the realm of atoms. Since then, quantum behaviour has been achieved with solid-state systems at the micro- and millimeter scale. Several solid-state systems show promise for the manufacturability of quantum processors with hundreds or thousands or more qubits.

Many exciting challenges lie ahead of realizing this promise on-chip, including materials, reliable fabrication, and connectivity between quantum elements. Some people say it might take another 20 years, I personally think it will be no more than 5 years…!

TECHNICAL UNIVERSITY OF DELFT (TUDELFT)
TU Delft says about Quantum Internet and Networked Computing: Our goal is to build an optically-connected network of many (small) quantum computers. Such a network enables the exchange of quantum bits between any of the connected quantum processors in order to solve problems that are intractable classically.
A quantum network in which the processors are located at different geographical locations is called a quantum Internet. Our goal is to develop the technology to enable quantum communication between any two places on earth. One application of such a quantum internet is to provide a fundamentally secure way of communication in which privacy is guaranteed by the laws of physics.
Quantum processors can also be connected into a quantum network in order to assemble a large quantum computing cluster. This approach is called networked quantum computing and offers a natural path towards scalability. Combining a quantum internet and a networked quantum computer finally allows remote users/providers to perform secure quantum computing "in the cloud".

Figure 9 / 10: some fantastic insight in the technical aspects of their lab

CONCLUSION
The **TUDelft** has received an enormous sum for research on their field and have a large commercial partner: **GOOGLE.**
So it seems a very feasible project… So the boolean statement will not be "Maybe Yes".

barnsten

- development tools
- consultancy
- components
- training
- hands-on workshops
- support

RAD Studio 10.2 Tokyo June 2017 Promotions BUY ONE, GET ONE

Rad Studio, C++ Builder or Delphi

Edition:	Professional	Enterprise	Architect
	⬇	⬇	⬇
Bogo Items	**Mobile Add-on Pack - 1 Year License** **- OR -** **InterBase Desktop 20 Desktop 20 License Pack**	**RAD Server 10 User License** **- OR -** **InterBase Desktop 100 License Pack**	**RAD Server Site License** **- OR -** **InterBase Desktop 100 License Pack**

Purchase now a RAD Studio, C++Builder or Delphi license.
You can download a free second product after the registration of your license.
You will receive the instructions with the delivery of your license(s).
The above table indicates which free product you can choose.

starter expert

Delphi & Lazarus

This article describes a Delphi project to calculate currents and voltages in complex DC networks of resistors and voltages sources. It allows also for the drawing, modification, storage and retrieval of networks. As an option, intermediate results may be observed while the process advances step by step.

Kirchhoff's circuit laws are two equalities that deal with the current and potential difference (commonly known as voltage) in the lumped element model of electrical circuits. They were first described in 1845 by German physicist **Gustav Kirchhoff.** *This generalized the work of* **Georg Ohm** *and preceded the work of* **Maxwell.** *Widely used in electrical engineering, they are also called* Kirchhoff's rules *or simply* Kirchhoff's laws.

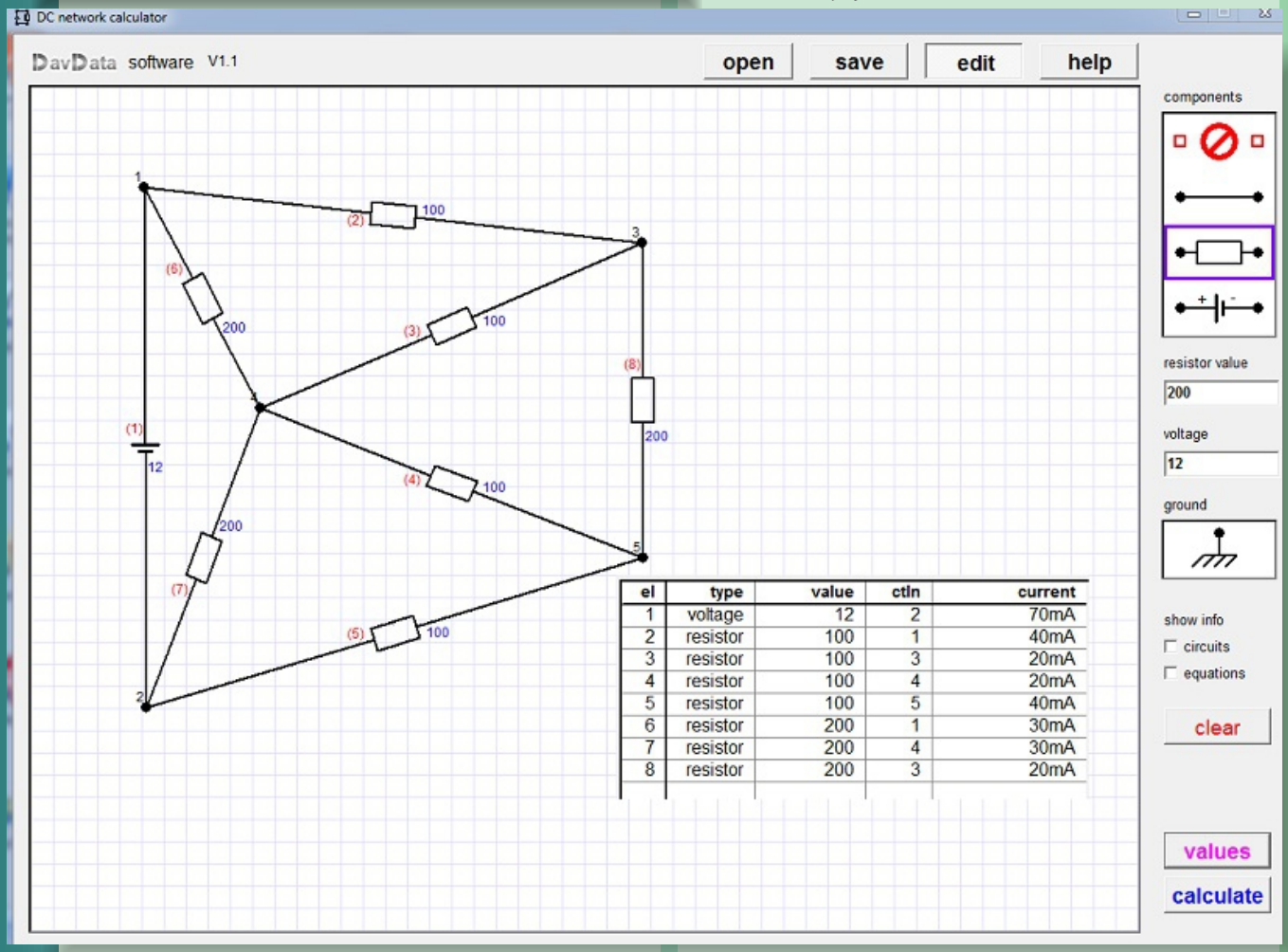


Figure 1: The running application

The program obtaines its results from the application of the laws of Kirchhoff.
These involve
1. the **current law**, and
2. the **voltage law** in electrical circuits which make a set of linear equations to be solved by Gauss-Jordan elimination.
Knowing the current in each element we work our way in the network starting at the 0 potential ground contact.
Using Ohm's law, the voltage of each interconnecting contact is calculated.

For those who love to see things explained we here have made use of **Wikimedia** to explain

Both of **Kirchhoff's laws** *can be understood as corollaries of the Maxwell equations in the low-frequency limit. They are accurate for DC circuits, and for AC circuits at frequencies where the wavelengths of electromagnetic radiation are very large compared to the circuits.*

KIRCHHOFF'S CURRENT LAW (KCL)
This law is also called Kirchhoff's first law, Kirchhoff's point rule, or Kirchhoff's junction rule (or nodal rule). The principle of conservation of electric charge implies that:
At any node (junction) in an electrical circuit, the sum of currents flowing into that node is equal to the sum of currents flowing out of that node
or equivalently

The algebraic sum of currents in a network of conductors meeting at a point is zero.
Recalling that current is a signed (positive or negative) quantity reflecting direction towards or away from a node, this principle can be stated as:

$$\sum_{k=1}^{n} I_k = 0$$

n *is the total number of branches with currents flowing towards or away from the node. See figure 2*

Figure 2

$$\sum_{k=1}^{n} \tilde{I}_k = 0$$

Figure 3

This formula is valid for complex currents: The law is based on the conservation of charge whereby the charge (measured in coulombs) is the product of the current (in amperes) and the time (in seconds). (See figure 3)

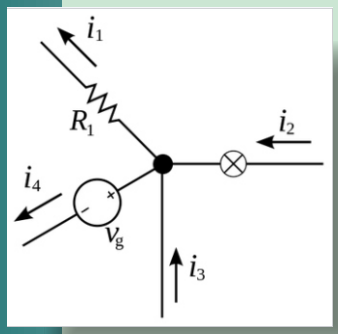


Figure 4:
The current entering any junction is equal to the current leaving that junction.

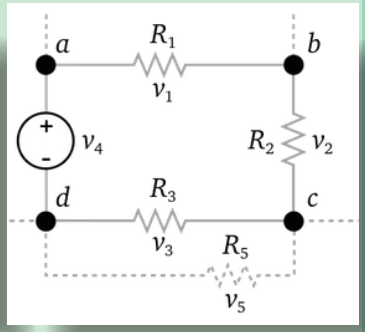


Figure 5:
The sum of all the voltages around a loop is equal to zero.

KIRCHHOFF'S VOLTAGE LAW (KVL)

This law is also called Kirchhoff's second law, Kirchhoff's loop (or mesh) rule, and Kirchhoff's second rule. The principle of conservation of energy implies that the directed sum of the electrical potential differences (voltage) around any closed network is zero, or: More simply, the sum of the emfs in any closed loop is equivalent to the sum of the potential drops in that loop, or: The algebraic sum of the products of the resistances of the conductors and the currents in them in a closed loop is equal to the total emf available in that loop. Similar to KCL, it can be stated as: *See figure 6*

$$\sum_{k=1}^{n} V_k = 0$$

Figure 6

Here, **n** *is the total number of voltages measured. The voltages may also be complex: See figure 7.*

$$\sum_{k=1}^{n} \tilde{V}_k = 0$$

Figure 7:

This law is based on the conservation of energy whereby voltage is defined as the energy per unit charge. The total amount of energy gained per unit charge must be equal to the amount of energy lost per unit charge, as energy and charge are both conserved.

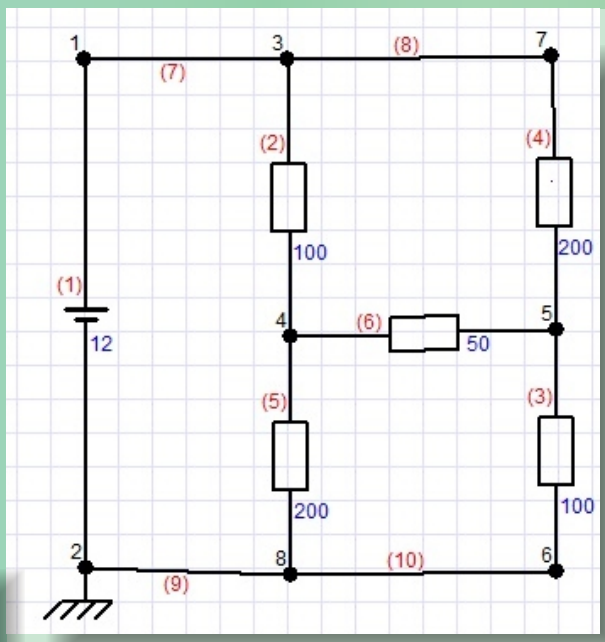


Figure 8: This image shows the dc-networks program at work

Element numbers are listed in (red). Values are written in blue. Elements can be interconnecting wires, resistors or dc voltage sources.
In the table 1 below:

```
ctIn     : contact in.
ctOut    : contact out
el       : element number
```

el	type	value	ctIn	current	ctOut		contact	voltage
1	voltage	12	2	87.273mA	1		1	12
2	resistor	100	3	54.545mA	4		2	0
3	resistor	100	5	54.545mA	6		3	12
4	resistor	200	7	32.727mA	5		4	6.545
5	resistor	200	4	32.727mA	8		5	5.455
6	resistor	50	4	21.818mA	5		6	0
7	wire		1	87.273mA	3		7	12
8	wire		3	32.727mA	7		8	0
9	wire		8	87.273mA	2			
10	wire		6	54.545mA	8			

Figure 9: Table 1

THE KIRCHHOFF CURRENT LAW

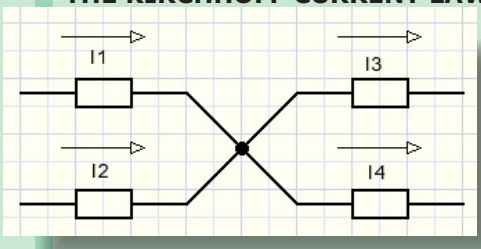


Figure 10: Table 1

We see 4 connected resistors with their currents I1..I4 The current law states that in any point of a network the sum of currents is zero (*no current can disappear*) so `I1 + I2 = I3 + I4` or

`I1 + I2 - I3 - I4 = 0`:

Each interconnecting point (contact) yields an equation. Written in the matrix form as known from linear algebra

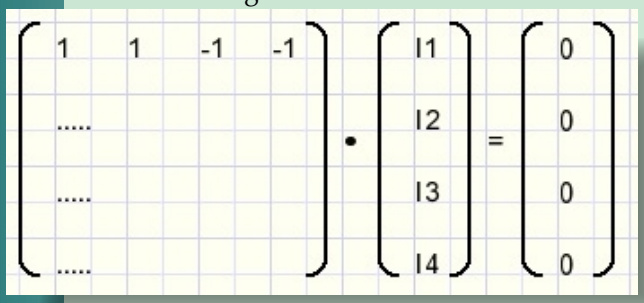


Figure 11: Matrix 1

THE KIRCHHOFF VOLTAGE LAW

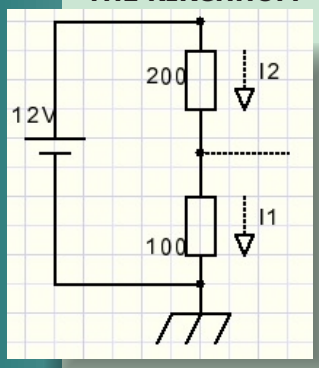


Figure 12: The circuit

Completing a round trip in the circuit, the sum of all voltages must be zero:

100I1 + 200I2 – 12 = 0 or
100I1 + 200I2 = 12

See matrix 2.
In matrix format:

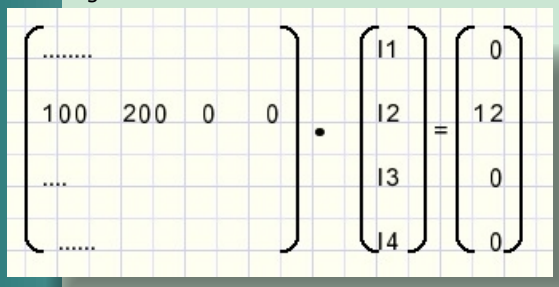


Figure 13: Matrix 2

For each compoment (*resistor or voltage source*) the smallest circuit *(loop)* is found. Each circuit again yields an equation. The total number of equations therefore is the sum of the number of interconnecting contacts and the number of resistors and voltage sources. Some equations may be redundant. To solve a network of n currents we need **n** independent equations

GAUSS-JORDAN ELIMINATION*

To solve the system of equations to find the value of currents I1..I4 we apply **Gauss-Jordan** elimination.
As a result, we obtain a matrix with all zeros except for the diagonal.

In the right column you see the final matrix 3 where all currents I1..I4 are solved.

** In linear algebra, Gaussian elimination (also known as row reduction) is an algorithm for solving systems of linear equations. It is usually understood as a sequence of operations performed on the corresponding matrix of coefficients. This method can also be used to find the rank of a matrix, to calculate the determinant of a matrix, and to calculate the inverse of an invertible square matrix. (Wiki)*

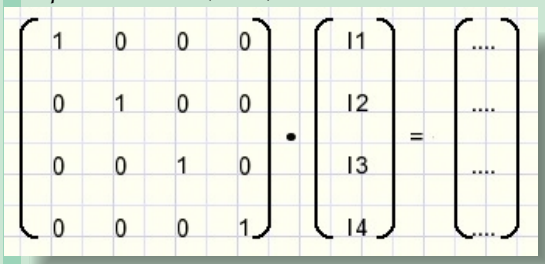


Figure 14: Matrix 3

THE DELPHI (7) PROJECT

The project has following forms and units

form1
menu buttons
paintbox to display the network
paintboxes showing components to be selected
edit boxes to enter component values (*resistance, voltage*)
label for messages

unit1
procedures for program control event handlers.
component paint procedures
Actually all painting is done in bitmaps.
Paintboxes only show the contents of these bitmaps.

network_unit
data formats and procedures
calculation of currents and voltages

resultform
paintbox to show table with results

result_unit
procedures to write the table.

I/O unit
open- and save procedures to load or save networks to disc.
Debugform, debug_unit
paintbox and procedures to show intermediate results
program execution may be paused to proceed step by step.

This explanation focusses on the network unit. Painting procedures, program control and I/O are not covered here.

DATA FORMATS

```
const maxelement = 30;
   maxcontact = 60;
   maxEC    = maxElement + maxContact;

type TElementType =
(etNone,etDelete,etWire,etResistor,etVoltage);
//the components, also called elements
   TS6 = string[6];
   TElement = record
         elType : TElementtype;
         con1   : byte;      // nr of interconnection
         con2   : byte;      //..
         value  : double; // Ohm, voltage
         vtext  : TS6;       // as entered in edit box
        end;
   TContact = record
         inUse : boolean;
         x,y : smallInt;   // coordinate on screen
        end;
var element : array[1..maxelement] of Telement;
  elCount : byte;          // number of elements in use
  contact : array[1..maxcontact] of TContact;
  groundContact : byte; // number of the ground contact
```

We notice elements (*wire, resistor, voltage source*) and contacts connecting the components.
The etDelete component is a dummy, painted to erase an existing component.

Within components we assume the current to flow from contact con1 to con2.
So a negative current flows from con2 to con1.

NOTE :
with elements and components I mean the same.
Elements are registered in array element[...].
New elements are added at the top of this array.
Deleting an element causes the higher elements in the array to shift down which decreases their number.

Contacts behave differently. They keep the same number in the contact[..] array if other contacts are deleted. So each entry of contact[...] needs a field "inUse" which is true if the contact is used.
Reason is that many elements may share a contact.

"Groundcontact" is the contact number that has the ground attached and is 0 volt.
If no ground is attached the value of groundcontact is zero.

THE EQUATIONS

```
var EQA   : array[0..maxelement,1..maxEC] of double;
// equation array
  topEQA : byte;  // number of equations
```

A contact results in an equation such as
I1 + I2 - I3 - I4 = 0
The I1 value is entered in **EQA[1,topEQA] := 1**
The I4 value is entered in **EQU[4,topEQA] := -1**
The 0 value right of =
is entered in **EQA[0, topEQA] := 0** (*the voltage*)

Which elements are connected to a certain contact?

```
var ECV : array[1..maxcontact] of dword;
// element-contact-bit vector
```

If element 10 is connected to contact 5 then `ECV[5]` has bit 10 set. `ECV` is used by procedure **GetNextFreeElement**, see later.

MAKING THE EQA[ ] AND ECV[..] ARRAYS

```
procedure makeEQA;  // make equation array
var i,j,n,nr : byte;
  mask : dword;
  mf : boolean;          // modify flag
 .....
begin //make ECV : bit set for element connected to contact
 for j := 1 to maxcontact do ECV[j] := 0;
 for i := 1 to elCount do
 with element[i] do
 begin
  mask := 1 shl i;
  ECV[con1] := ECV[con1] or mask;
  ECV[con2] := ECV[con2] or mask;
 end;
.....
```

KIRCHHOFF'S CURRENT LAW

```
procedure makeEQA;  // make equation array
.............see before.........

// Kirchhoff currents

 for j := 1 to maxContact do   // scan all contacts
 begin
  nr := topEQA + 1;
  mf := false;
  for i := 1 to elCount do       // Kirchhoff current
  with element[i] do
   begin
    if j = con1 then begin
              EQA[i,nr] := 1;
              mf := true;
             end;
    if j = con2 then begin
              EQA[i,nr] := -1;
              mf := true;
             end;
   end; // for i
  if mf then topEQA := nr;
 end;  // for j
.....continued.....
```

Well, this was the easy part.

FINDING CIRCUITS

A circuit is a path starting at a contact of an element , through other elements and returning to the original contact. The sum of voltage drops across elements must be zero.

Data structures:

```
type Tcircuit = record
        ctIn,ctOut : byte;     // entry,exit contact
        el : byte;             // element
        mult : shortInt;       // 1, -1 direction multiplier
       end;

var circuit : array[1..maxContact] of Tcircuit;
  ccLength: byte;              // entries in circuit array
  ECV   : array[1..maxcontact] of dword;     // element-contact-vector
  CEF   : array[1..maxcontact] of boolean; // contact enable flag
  EEF   : array[1..maxelement] of boolean; // element enable flag
  CCOK  : boolean;             // circuit OK
```

The boolean array CEF[ ] enables contacts. A false flag prevents that the contact is re-selected. Array **EEF[ ]** enables elements and prevents re-election of an element already part of a circuit. The array circuit[ ] holds the elements that make a circuit. Variable mult (*sign multiplier*) equals 1 if **ctIn = con1** and **ctOut = con2.mult = -1 if ctIn = con2 and ctOut = con1**, when current flow is reversed. Remember that current is assumed to flow from **element[ ].con1** to **element[ ].con2** contacts.

One of the **arrays CEF, EEF** seems superfluous. When searching for a circuit (loop) it is sufficient not to select contacts that are already part of the circuit. However, the last contact in a circuit is also the starting contact. This contact must be choosen as the last one. Elements may never appear twice in a circuit.

So I record both elements and contacts.

To save calculations, the smallest possible cicuit loop is used for each element.

This means that all possible circuits have to be examined and the smallest choosen. The minimal circuit length is 3 (*elements including wires*).

Because **array circuit[ ]** holds the shortest circuit of elements we need another array to search for a circuit. This array is called **node[ ]** of **Tcircuit.Node[ ]** is copied to **circuit[ ]** only if **node[ ]** is smaller in size or no circuit was found before. Array Node[ ] acts as a multi-digit counter where the elements are the digits. By systematically advan - cing this counter while testing for a round trip , circuits are generated. So this is like a "brute force" search, but care is taken not to use a contact or element twice

NOTE:

"Node" is obtained from graph theory.

EXAMPLE

To explain the procedures that make and manipulate the **EQA[ ]** equations array please look at the next simple diagram:

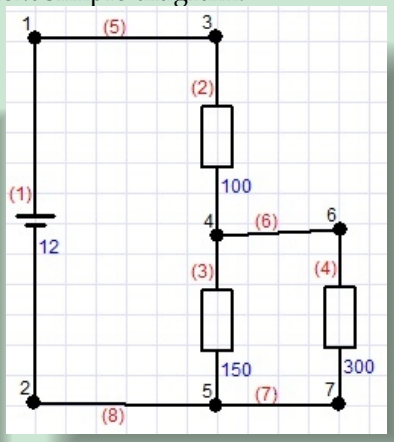


Figure 15: Diagram

The voltage source is 12V, resistor values are 100, 150, 300 Ohm. At the end, these are the calculated currents

el	type	value	ctIn	current	ctOut
1	voltage	12	2	60mA	1
2	resistor	100	3	60mA	4
3	resistor	150	4	40mA	5
4	resistor	300	6	20mA	7
5	wire		1	60mA	3
6	wire		4	20mA	6
7	wire		7	20mA	5
8	wire		5	60mA	2

Figure 16: Overview of the calculated currents

The generated circuits (*loops*) for Kirchhoff's voltage law are:

nr	ct	el	ct	el	ct	el	ct	el	ct	el
1	(1)	1	(2)	8	(5)	3	(4)	2	(3)	5
2	(3)	2	(4)	3	(5)	8	(2)	1	(1)	5
3	(4)	3	(5)	7	(7)	4	(6)	6		
4	(6)	4	(7)	7	(5)	3	(4)	6		

Figure 17: generated circuits

NOTE: ct=contact ; el=element

The equations array after applying the current- and the voltage laws:

I1	I2	I3	I4	I5	I6	I7	I8	=
1	0	0	0	1	0	0	0	0
-1	0	0	0	0	0	0	1	0
0	1	0	0	-1	0	0	0	0
0	-1	1	0	0	1	0	0	0
0	0	-1	0	0	0	1	-1	0
0	0	0	1	0	-1	0	0	0
0	0	0	-1	0	0	-1	0	0
0	-100	-150	0	0	0	0	0	12
0	100	150	0	0	0	0	0	-12
0	0	150	-300	0	0	0	0	0
0	0	-150	300	0	0	0	0	0

Figure 18: equation array

Note, that the first 7 equations are the result of the current law for contacts **1 . . 7**
The last four equations are the result of the voltage law starting at elements **1 . . 4**

Finally after Gauss-Jordan elimination we see the current through each element:

I1	I2	I3	I4	I5	I6	I7	I8	=	
1	0	0	0	0	0	0	0		0.06
0	1	0	0	0	0	0	0		-0.06
0	0	1	0	0	0	0	0		-0.04
0	0	0	1	0	0	0	0		-0.02
0	0	0	0	1	0	0	0		-0.06
0	0	0	0	0	1	0	0		-0.02
0	0	0	0	0	0	1	0		0.02
0	0	0	0	0	0	0	1		0.06
0	0	0	0	0	0	0	0		0
0	0	0	0	0	0	0	0		0
0	0	0	0	0	0	0	0		0
0	0	0	0	0	0	0	0		0

Figure 19: current through each element:

As we saw, procedure makeEQA is called to make the equation array from the Kirchhoff laws.
Finding the shortest circuit starting from each voltage- or resistor element is done by function `findcircuit(fel : byte) : boolean;`
fel is the first element.
Findcircuit exits true if the circuit loop is found.

Function findcircuit starts by setting `CEF[ ]` and `EEF[ ]` true for all contacts and all elements. Then the `node[ ] array` is cleared which is not really necessary but it helped the debugging process. As a last initializing step, `node[1]` is preset:

```
with node[1] do        // set 1st node
 begin
  ctIn := element[fel].con1;
  ctOut := element[fel].con2;
  el := fel;
  mult := 1;
  CEF[ctOut] := false;
 end;
EEF[fel] := false;
nn := 1;
```

`CEF flag` is set false for the output contact.
EEF[ ] is set false for the element.
This avoids selecting this contact and element again. **nn** is the node number.

The next part of `function findcircuit( )` is a loop build with labels and goto statements.
Label nextNode: incrementing nn selects the next node. The input contact is set to the output contact of the previous node. The element is set to zero, no element is selected yet.

`Label NextMove:`
if an element exists already, it is removed by re-enabling this element and its output contact.
function `NextFreeElement(nel,ctIn )` is called which returns the next element nel connected to contact ctIn. Mult is set to 1 or -1 depending on the current flow

Then a check is made for a circuit loop:
`ctOut = node[1].ctIn)`
If the loop in node[ ] is shorter then in `circuit[ ]` then `circuit[ ]` is replaced by `node[ ]`
If no circuit loop then goto `NextNode;`

The last part of the function could be labeled "previousNode" but no label is necessary.
Only if the node number is 3 or higher we return to the previous node.
(**`node[1]`** *data is fixed and cannot be updated*)
The current node is closed which means
re-enabling `CEF[ctOut]`
re-enabling `EEF[el]`
continue at label `nextMove;`

As a last step:
what is function `getNextFreeElement(var el: byte; ct:byte) : boolean ;` doing?
A free element is found which is higher then el and connected to contact ct.

This makes the nodes act as a counter of the elements.
A while loop scans the elements and contacts.
If an element is found it is returned in var el and also the EEF[ ] and CEF[ ] arrays are set false for the contact and element found.

SOLVING THE EQA ARRAY
This is done by the Gauss-Jordan elimination which is a simple and systematic process.
Procedure solveEQA; does the job.
This procedure has four parts:

1. Gausss-Jordan down
2. Gauss-Jordan up
3. normalize
4. short circuit check
5. voltage list

Gauss-Jordan down
In a system of equations an equation may be multiplied by a constant and also equations may be added, subtracted.

EQA[i,n] is the value at row n and column i.
Starting at column 1 the first row if found that has a non zero value in this column.
If found, rows i and n are swapped.
Now, rows i and j (j starting at i+1) are compared for column i.
variable `M = EQA[i,j] / EQA[i,i] ;` M is multiplier.
`EQA[i,j] = 0` and for the other columns:
`EQA[k,j] := EQA[k,j] - M*EQA[k,i] ;`
In this way we sweep column i to zero starting from row i.

Gauss-Jordan up
Now we start at the highest element (or column) and use the same method as before but now we sweep column i to zero above row i.

Note: both in the up and down procedures very small values resulting from floating point rounding are set to zero.
At this point the array EQA has all zero value except for the diagonal [i,i]

Normalize
The [i,i] values are normalized to 1 by division.

```
for i := 1 to elCount do
 if EQA[i,i] <> 0 then
 begin
  EQA[0,i] := EQA[0,i] / EQA[i,i];
  EQA[i,i] := 1;
 end
  else CCOK := false;
```

Remember that the voltage valus are stored in **EQA[0,i]**

Short circuit check
To solve the currents through elCount elements we need elCount (independent) equations.
Initially there could have been many more equations than that.
EQA[0,n] for n > elCount must be zero because an equation like:
0*I1 + 0*I2 + 0*I3 + 0*I4 = 15
is impossible. (*results from a short cicuit*)

Voltage list

```
type TVolt = record
       ok : boolean; // voltage calculated
       v : double;
      end;
var CVA : array[1..maxcontact] of Tvolt;
                    // voltages per contact
```

This part is executed only if the ground strap is attached (*groundcontact* > *0*)
A scan through all elements is repeated as long as changes are made (*boolean variable nochange* = *false)*
When the voltage of a contact is known the voltage of the other contact is calculated.

This concludes the description of the dcnetwork project.

For details, please refer to the (**DELPHI-7**) source code.
This means that the project is updatable to any newer version of Delphi and to Lazarus

About the author
David Dirkse
After a study of electrical engineering, David joined the Control Data Corporation, an American computer company, and worked almost 25 years installing and maintaining scientific data centers in the Netherlands. He has witnessed the evolution of computer hardware from the first transistorized mainframes (CDC3300) to the supercomputers of the eighties (CYBER205). During this period he attended over 100 technical courses and was instructor in over 30 occasions. At the decline of CDC around 1990, David started a study of mathematics and became a math teacher in 1993. One of his hobbies is programming, specializing in puzzle solving, drawing and basic algorithms.

BY DETLEF OVERBEEK

starter expert DX Delphi

Abstract:
This is the only version of a MemTable I know of that can work with SQL. It really does! And I think that is a big advantage.

Introduction:
In this project we will use the MemTable of kbmMW, (Components4Developers) created by Kim Madsen. To make use of the project which includes the SQL support you will need to use at least the kbmMW CodeGear Edition of the kbmMemTable which is free. The professional edition costs only $35 and you certainly get value for money. But if you want a standalone kbmMemTable without kbmMW, they need to purchase kbmMemTable Standard Edition.

There is one thing: if you want to customize your program, you will have to invest some extra time to make your idea work. There is really a lot you can do using this SQL version, but there is a learning curve before you can proficiently exploit its power. At the end of the article there is a project with full code you can download where I show how you could do this. So this article is divided into two parts, firstly the MemTable project and secondly the demo project where you can find all sorts of solutions for SQL use.

PART 1.

CREATING THE PROJECT:
As we did in the other two projects (*Blaise 60, about ClientDatasets; Blaise 61, about ClientDatasets from FireDac*) we will use the same template.

Only this time we will clean up all other components before we add new ones from **kbmMW.**
First of all something about the installation of this suite.
If you already had an earlier version or the Embarcadero version, I urge you to remove that very carefully and then again install it.
Read the installation documents,
there is an `installation.text.`
It's not difficult but you must do it step by step.
It installs under:
`c:\Program Files (x86)\kbmMemTable\.`
There is a **Demo directory** and a **SQL Demo** Directory. Which is divided into **VCL** and **FMX.**
There is also a help directory that contains an older format of the help file(*pre Windows 7*).
I used it and that works quite good although the help is not very helpful or comprehesive. There is also a PDF-Help file that you can download, which has identical help content.
The help itself has the same format as we were used to under Delphi7. That is no longer supported by Windows, but it can be used as a separate help to open.

So F1 brings you only to the Delphi dataset help environment.

In this version, which is in itself not different to the other versions I wrote of before, we have one thing that is very special:
The `kbmMemSQL.table.` By using it as table replacing a MemTable you are allowed to make use of the SQL possibilities it has:

`SELECT, INSERT, UPDATE` and **`DELETE`**
The component executing the SQL is named `TkbmMemSQL.`
It supports registering multiple **`kbmMemTables`** with it which each can be aliased
(*as shown in the demo in the* **ToDo** *app*) shows the component kbmMemTable1 is aliased as table1.

It supports:
complex calculations,
`MOD, DIV, +, -, *, /, (, ),`
`AS aliasing, LIKE,`
`BETWEEN, IN, <,>,<=,>=,<>, NOT,`
`SELECT,`
`DELETE, UPDATE, INSERT, IS NULL,`
`IS NOT NULL, ORDER BY,`
`DESC, GROUP BY, MAX, MIN, SUM,`
`AVG, COUNT`

Not (yet) supported:
`(HAVING is supported, HAVING works along with GROUP BY), but no sub selects yet, other DDL commands, joins.`

Here is a brief overview of second Project giving you a more explicit explanation of what can be done.

For our **TODO** demo I wrote some simple lines of SQL so that you even beginners can understand what is going on.
If you want refreshing about use of the buttons and of the code how to load a DataSet etc.
I suggest you read the two earlier Memtable / ClienDataSet articles in issue 60 and 61.
So to start **Part1** take a look a Figure1.
Here is an overview of all the components and you can get a good overview of what differs from the two earlier Client Dataset versions.
Have a good look at all the buttons and then the User Interface will become more understandable.

To make the app not too large onscreen I have chosen not to have a lot of extra forms and they are loaded on top of each other when you use it.

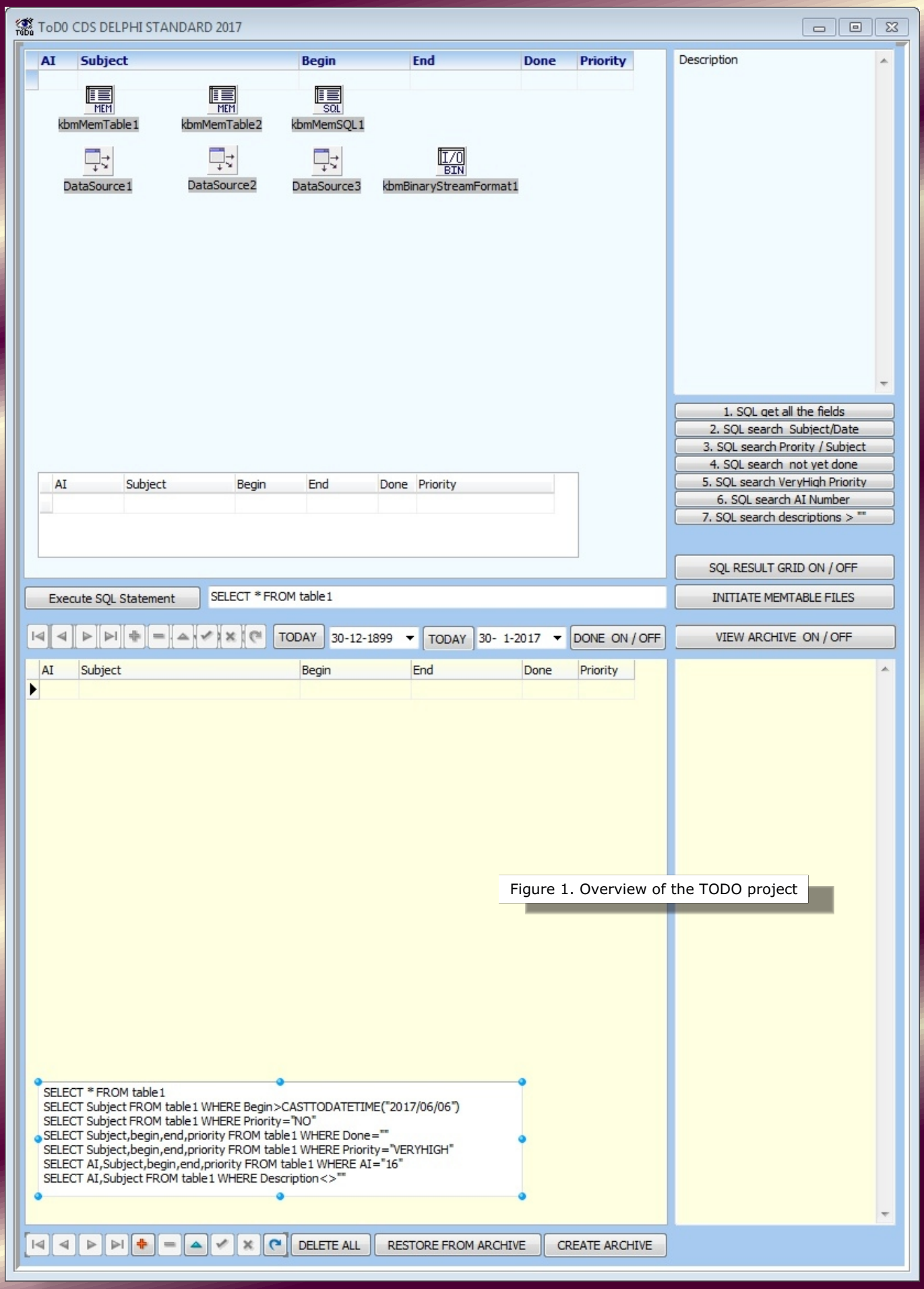


Figure 1. Overview of the TODO project

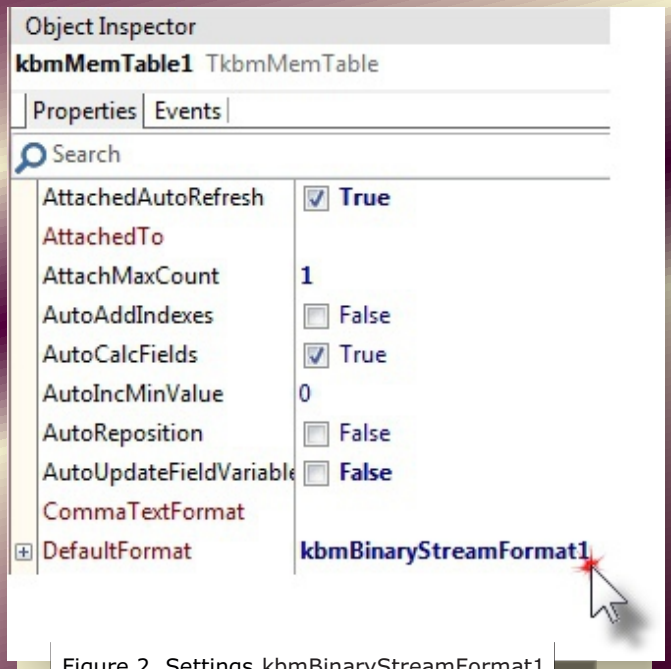


Figure 2. Settings kbmBinaryStreamFormat1

In this overview of Figure 1 you can see the design-time form:
In the blue background there is a `DBGrid1` which is connected to `Datsource1`, and `Datsource1` is connected to `kbmMemTable1`.
In the **Object Inspector** of the `kbmMemTable1`find can find the `Default Format property`, and must set it to `kbmBinaryStreamFormat1`.
It will then appear in your drop down choice.
There is also a `CommaTextFormat`, a format that might be very handy for future purposes.
Other MemTables don't have this option.

The `kbmBinaryStreamFormat1` is also placed on the form. I didn't use the `CommaTextFormat`. `KbmBinaryStreamFormat1` is used for creating the files where the data are stored for this project.
Under the button **INITIATE MEMTABLE FILES** is this event code:

```
procedure TF_TODO.Button4Click(Sender: TObject);
begin
 kbmMemTable2.SaveToFile(ExtractFilePath(Application.exename)+'ToDoList_kbmMW_Binary.detlef');
 kbmMemTable2.SaveToFile(ExtractFilePath(Application.exename)+'ToDoListBackUp_Binary.detlef');
end;
```

Of course you can rename the files, be aware that there are two other places where you need to rename.
The first:

```
procedure TF_TODO.FormCreate(Sender: TObject);
begin

 kbmMemsql1.Parser.FormatSettings.DateSeparator:='/';
 kbmMemsql1.Parser.FormatSettings.ShortDateFormat:='yyyy/mm/dd';
 F_Todo.Height := 604;
 DBGrid3.Left  := 2;
 DBGrid3.Top   := 2;
 DBGrid3.Height := 516;
 DBGrid3.Width := 585 ;
 DateTimePicker1.DateTime := (Now);
 DateTimePicker2.DateTime := (Now);
 kbmMemTable1.Open;
 kbmMemTable1.Active;
 kbmMemTable2.Open;
// Loading
 kbmMemTable1.LoadFromFile(ExtractFilePath(Application.exename)+'ToDoList_kbmMW_Binary.detlef');
 kbmMemTable2.LoadFromFile(ExtractFilePath(Application.exename)+'ToDoListBackUp_Binary.detlef');
 ReportMemoryLeaksOnShutdown:=true;

end;
```

`kbmMemTable1.LoadFromFile` loads the files

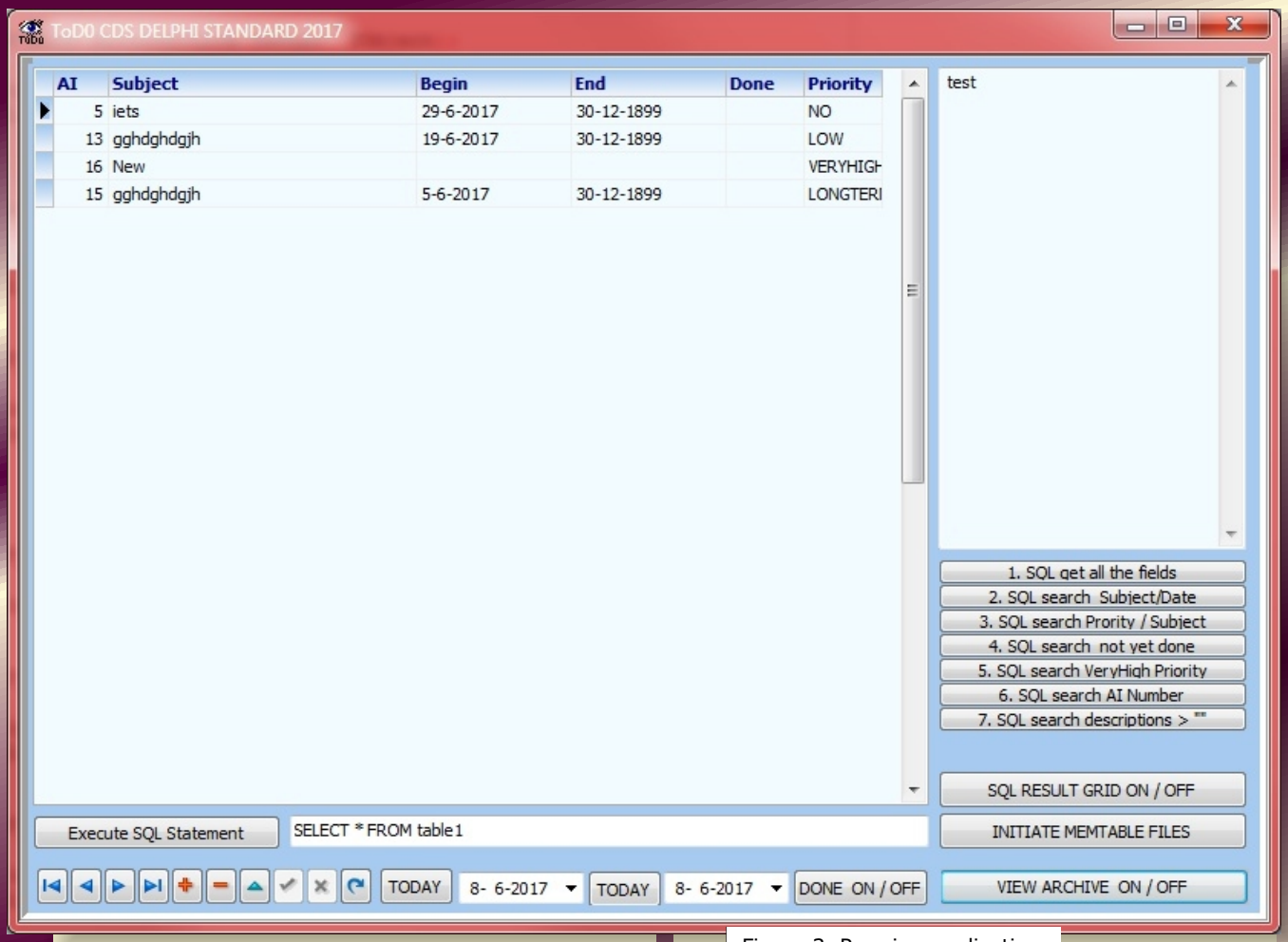


Figure 3. Running application

```
procedure TF_TODO.FormClose(Sender: TObject; var Action: TCloseAction);
begin

 If (kbmMemTable1.State = dsEdit, dsInsert) then  kbmMemTable1.Post;
 kbmMemTable1.saveToFile(ExtractFilePath(Application.exename)+'ToDoList_kbmMW_Binary.detlef');

 If (kbmMemTable2.State = dsEdit, dsInsert) then  kbmMemTable2.Post;
 kbmMemTable2.Post;
 kbmMemTable2.SaveToFile(ExtractFilePath(Application.exename)+'ToDoListBackUp_Binary.detlef');

end;
```

Second place: rename at the closing of the program is the second option to save the files `OnClose (FormClose)`. See code above

The "**INITIATE MEMTABLE FILES -** Button" needs to be used only once to create the files for the first time. After that you can make it invisible or simply remove it.

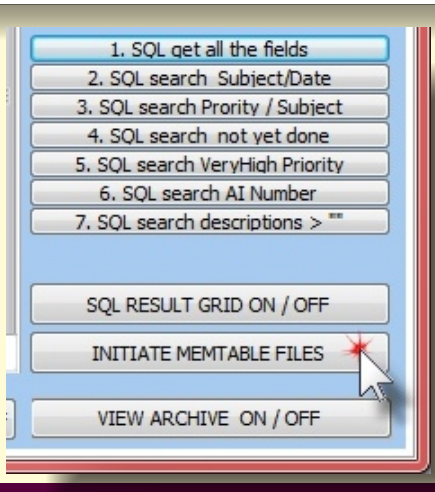


Figure 4. The group of SQL Buttons

There are some things to explain from the form create procedure:

```
kbmMemsql1.Parser.FormatSettings.DateSeparator:='/';
kbmMemsql1.Parser.FormatSettings.ShortDateFormat:='yyyy/mm/dd';
```

We need this to make sure that some SQL which uses the time or date format can work.
There is a final but very interesting tool which you can use.

```
ReportMemoryLeaksOnShutdown:=true;
```

Hopefully it does not find any leaks!

SQL WITH KBMMEMSQL1

I had the idea it should really be easy to create SQL scripts and except for some advanced uses it is: the seven buttons do simple jobs except for one:
Button "2 SQL Search Subject /Date":

```
SELECT Subject FROM table1 WHERE
Begin>CASTTODATETIME("2017/06/06")
```

After viewing the code it is quite self-explanatory. I have added the SQL code added to a memo file (mSQL), this is easy just for now.
Creating these SQL scripts is made possible by the following section of code:

```
procedure TF_TODO.ExecuteSQL(const ASQL:string);
begin
  // Add tables that the SQL is supposed to access.
  kbmMemSQL1.Tables.Clear;
  kbmMemSQL1.Tables.Add('table1',kbmMemTable1);
  kbmMemSQL1.ExecSQL(ASQL);
  DataSource3.DataSet:=kbmMemSQL1;
```

in this procedure you can see how the kbmMemSQL1 adds the kbmMemTable1 which then can be executed as follows

```
procedure TF_TODO.Button1Click(Sender: TObject);
begin
  DBGrid3.Visible := True;
  ExecuteSQL(mSQL.Lines[0]); // writes the sql

end;
```

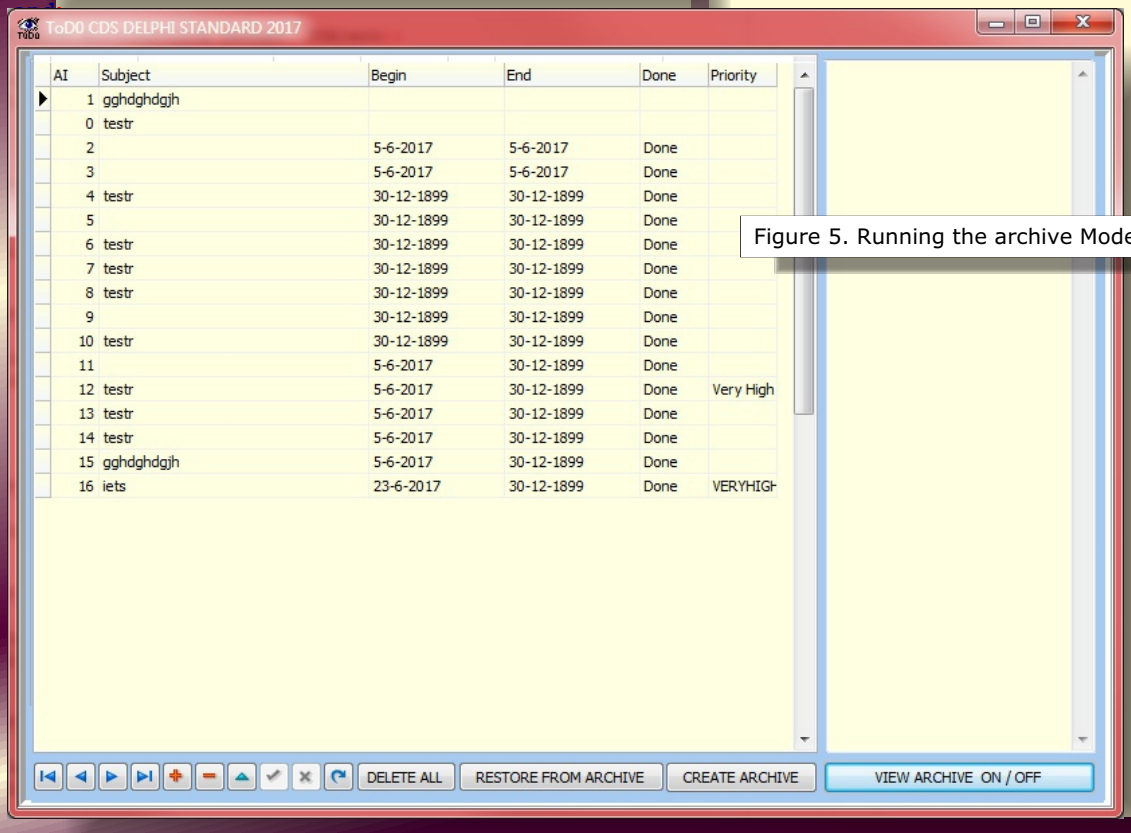


Figure 5. Running the archive Mode

Here is the complete list of all the SQL scripts:

```
SELECT * FROM table1
SELECT Subject FROM table1 WHERE Begin>CASTTODATETIME("2017/06/06")
SELECT Subject FROM table1 WHERE Priority="NO"
SELECT Subject,begin,end,priority FROM table1 WHERE Done=""
SELECT Subject,begin,end,priority FROM table1 WHERE Priority="VERYHIGH"
SELECT AI,Subject,begin,end,priority FROM table1 WHERE AI="16"
SELECT AI,Subject FROM table1 WHERE Description<>""
```

Since this code is added to the Memo (`MSQL.lines`): you'll have all that in one place. To make it easy I also made it possible to execute the SQL using an Edit field with the button: **Execute SQL Statement.** If you add your SQL to the button you simply click on it and it will be executed:
You could do this like this:

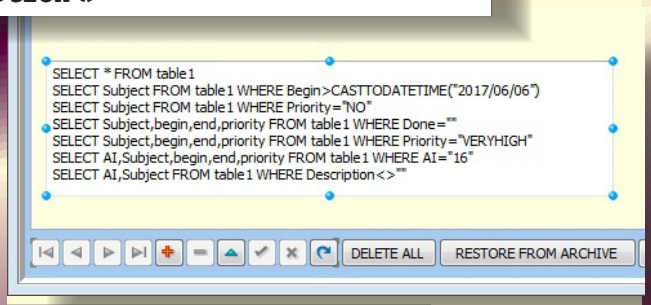


Figure 6. The not visible SQL Memo

```
procedure TF_TODO.BtnSQLExeClick(Sender: TObject);
Var S : String ;
begin
 S:= Edit1.Text;
 ExecuteSQL(S);
 DBGrid3.Visible := True;
end;
```

or for just clicking the edit field:

```
procedure TF_TODO.Edit1Click(Sender: TObject);
begin
 DBGrid3.Visible := True;
 ExecuteSQL(Edit1.Text);
end;
```

Figure 7. Search High Priority

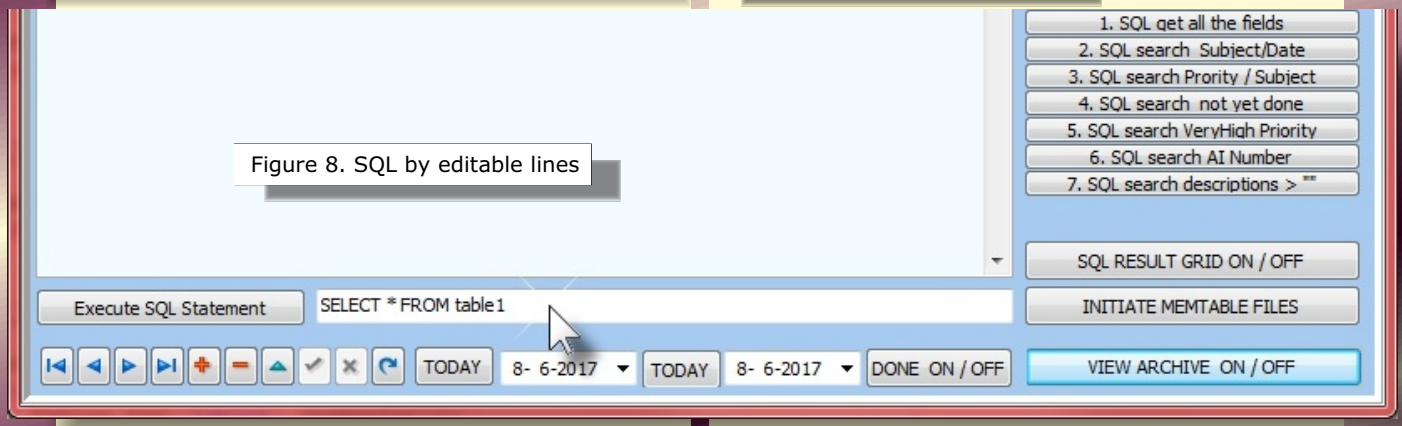


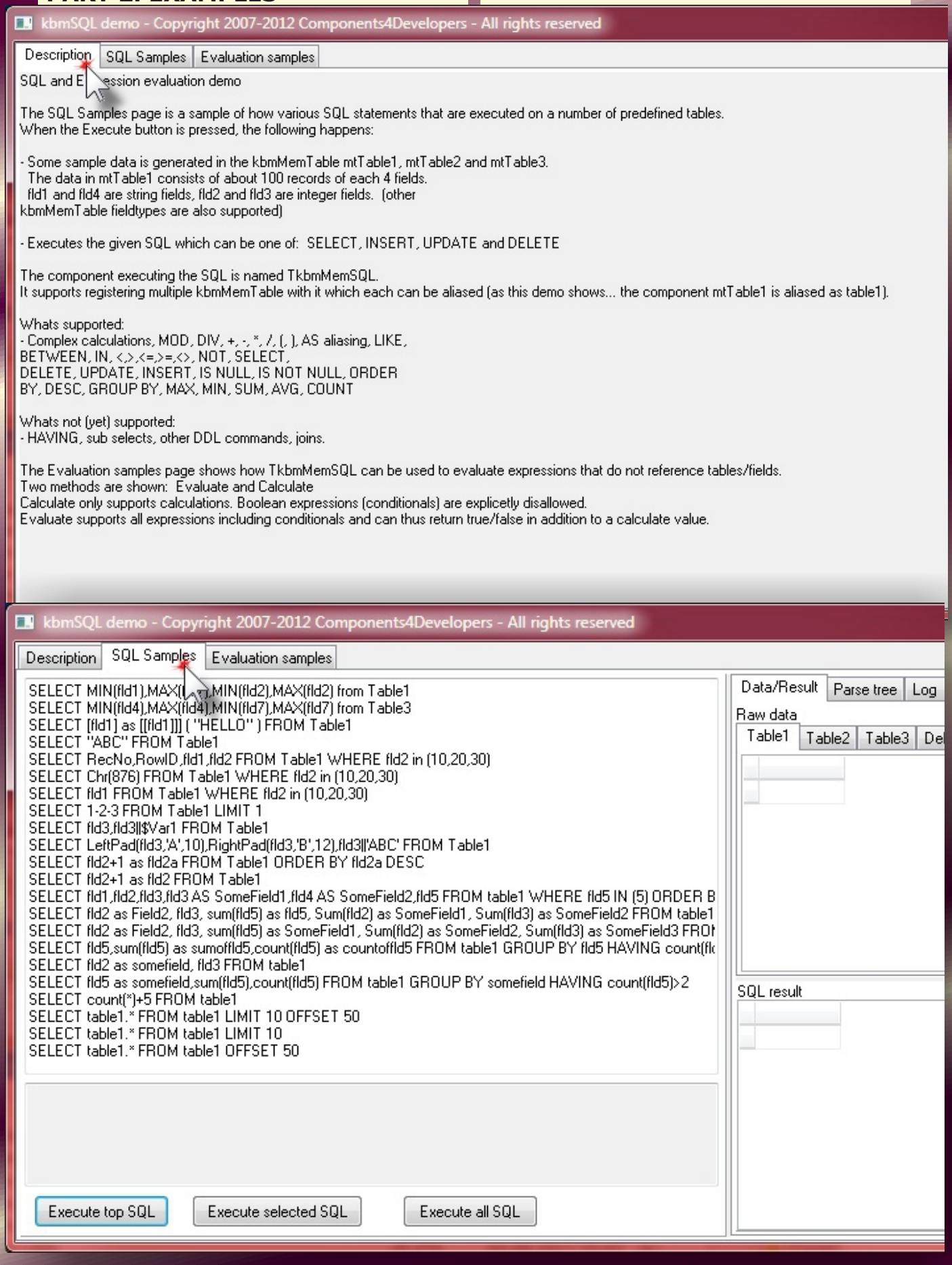
Figure 8. SQL by editable lines

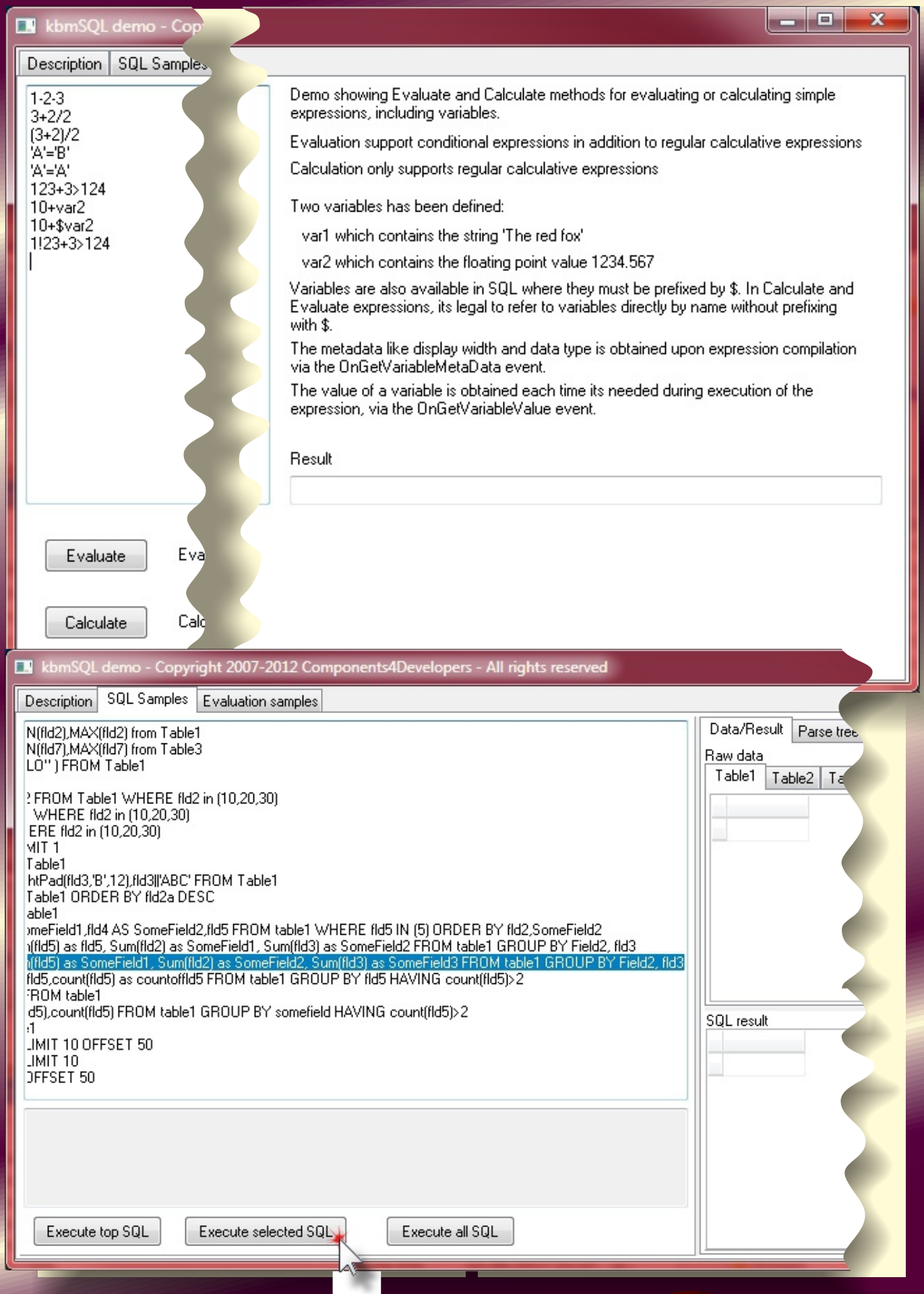
The complete code will be available at your personal download page:
http://www.blaisepascal.eu/loginnew.php
The other details like all the code of the buttons is available in the project .

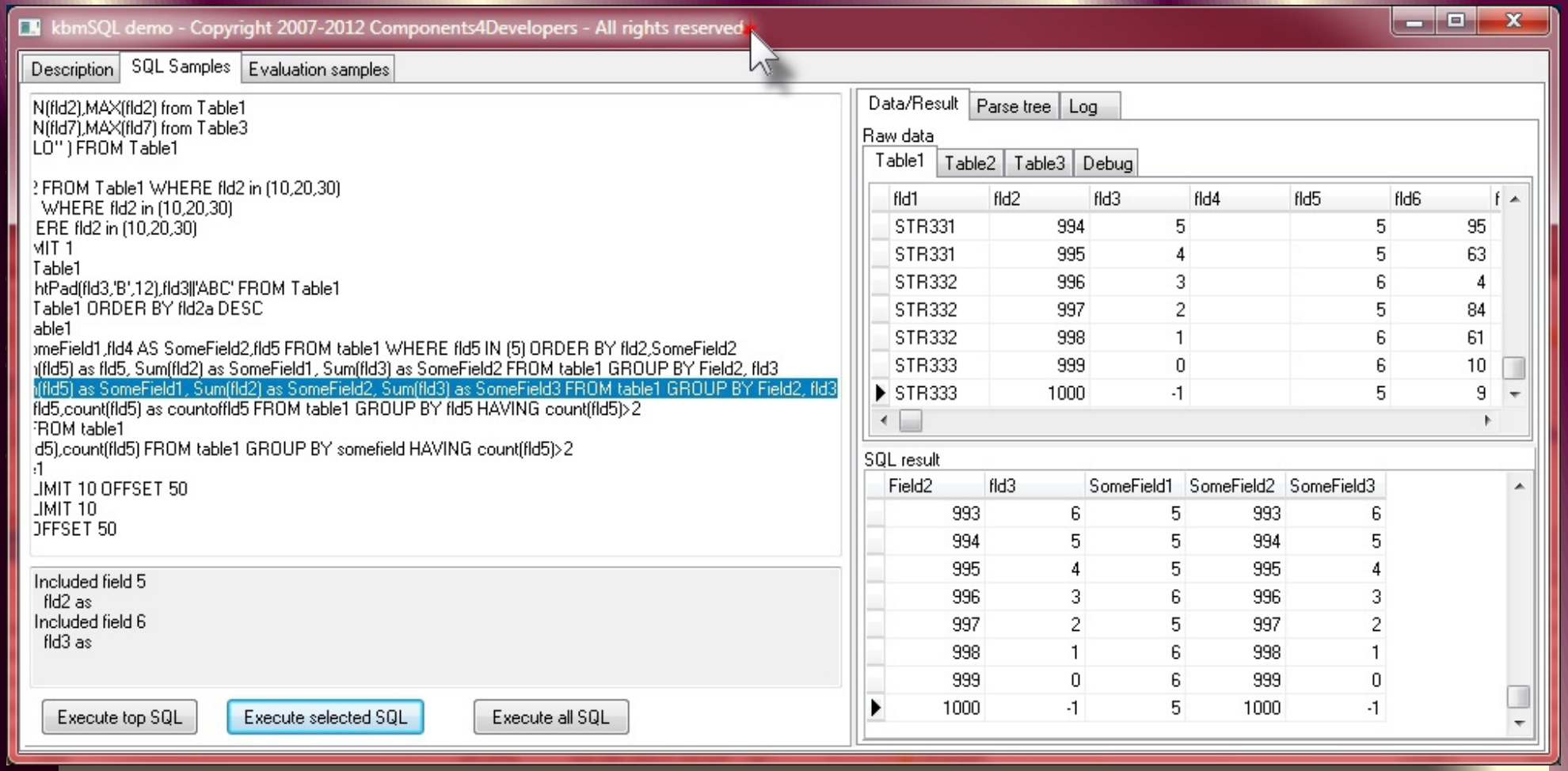
As an extra you can download the project of the Demo file Kim has written for you. This is also available at your download page. You will need kbmMW CodeGear Edition or kbmMemTable Standard Edition if you want to use the demo and this TODO project.

On the next pages there are some views available of that project (Part 2)

PART 2. EXAMPLES




kbmSQL demo - Cop
Description
SQL Samples
1-2-3
3+2/2
(3+2)/2
'A'='B'
'A'='A'
123+3>124
10+var2
10+$var2
1!23+3>124
Demo showing Evaluate and Calculate methods for evaluating or calculating simple expressions, including variables.
Evaluation support conditional expressions in addition to regular calculative expressions
Calculation only supports regular calculative expressions
Two variables has been defined:
var1 which contains the string 'The red fox'
var2 which contains the floating point value 1234.567
Variables are also available in SQL where they must be prefixed by $. In Calculate and Evaluate expressions, its legal to refer to variables directly by name without prefixing with $.
The metadata like display width and data type is obtained upon expression compilation via the OnGetVariableMetaData event.
The value of a variable is obtained each time its needed during execution of the expression, via the OnGetVariableValue event.
Result
Evaluate
Calculate
kbmSQL demo - Copyright 2007-2012 Components4Developers - All rights reserved
Description
SQL Samples
Evaluation samples
N(fld2),MAX(fld2) from Table1
N(fld7),MAX(fld7) from Table3
LO'') FROM Table1
FROM Table1 WHERE fld2 in (10,20,30)
WHERE fld2 in (10,20,30)
ERE fld2 in (10,20,30)
MIT 1
Table1
htPad(fld3,'B',12),fld3||'ABC' FROM Table1
Table1 ORDER BY fld2a DESC
able1
meField1,fld4 AS SomeField2,fld5 FROM table1 WHERE fld5 IN (5) ORDER BY fld2,SomeField2
(fld5) as fld5, Sum(fld2) as SomeField1, Sum(fld3) as SomeField2 FROM table1 GROUP BY Field2, fld3
(fld5) as SomeField1, Sum(fld2) as SomeField2, Sum(fld3) as SomeField3 FROM table1 GROUP BY Field2, fld3
fld5,count(fld5) as countoffld5 FROM table1 GROUP BY fld5 HAVING count(fld5)>2
ROM table1
d5),count(fld5) FROM table1 GROUP BY somefield HAVING count(fld5)>2
1
IMIT 10 OFFSET 50
IMIT 10
FFSET 50
Data/Result
Parse tree
Raw data
Table1
Table2
SQL result
Execute top SQL
Execute selected SQL
Execute all SQL


kbmSQL demo - Copyright 2007-2012 Components4Developers - All rights reserved
Description
SQL Samples
Evaluation samples
N(fld2),MAX(fld2) from Table1
N(fld7),MAX(fld7) from Table3
LO'') FROM Table1
: FROM Table1 WHERE fld2 in (10,20,30)
WHERE fld2 in (10,20,30)
ERE fld2 in (10,20,30)
MIT 1
Table1
htPad(fld3,'B',12),fld3||'ABC' FROM Table1
Table1 ORDER BY fld2a DESC
able1
meField1,fld4 AS SomeField2,fld5 FROM table1 WHERE fld5 IN (5) ORDER BY fld2,SomeField2
(fld5) as fld5, Sum(fld2) as SomeField1, Sum(fld3) as SomeField2 FROM table1 GROUP BY Field2, fld3
(fld5) as SomeField1, Sum(fld2) as SomeField2, Sum(fld3) as SomeField3 FROM table1 GROUP BY Field2, fld3
fld5,count(fld5) as countoffld5 FROM table1 GROUP BY fld5 HAVING count(fld5)>2
FROM table1
d5),count(fld5) FROM table1 GROUP BY somefield HAVING count(fld5)>2
:1
_IMIT 10 OFFSET 50
_IMIT 10
)FFSET 50
Included field 5
fld2 as
Included field 6
fld3 as
Execute top SQL
Execute selected SQL
Execute all SQL
Data/Result
Parse tree
Log
Raw data
Table1
Table2
Table3
Debug
fld1 fld2 fld3 fld4 fld5 fld6
STR331 994 5 5 95
STR331 995 4 5 63
STR332 996 3 6 4
STR332 997 2 5 84
STR332 998 1 6 61
STR333 999 0 6 10
STR333 1000 -1 5 9
SQL result
Field2 fld3 SomeField1 SomeField2 SomeField3
993 6 5 993 6
994 5 5 994 5
995 4 5 995 4
996 3 6 996 3
997 2 5 997 2
998 1 6 998 1
999 0 6 999 0
1000 -1 5 1000 -1


AWAITING YOUR COMMANDS
MASTER...
www.visuino.com
VISUINO

INTRODUCTION TO VIDEO PROCESSING: PAGE 1/10

BY BOIAN MITOV

starter expert

For many years,
Delphi was commonly considered a DB front-end Development Environment. It is true that Delphi really excels in this role, but Delphi is indeed very powerful native development platform, competing well, and often outperforming other platforms. Rarely it is more evident, than when developing Multimedia Applications. Implementing Video and Audio processing applications requires high performance native processing with no lag, and can easily push the system and the development environments to their limits. Delphi performs surprisingly well in this role, not only being able to process multiple video, and audio streams simultaneously. but even to perform computer vision in real time.

Windows for many years has been the dominant desktop operating system, and the primary platform supported by Delphi. When I started developing Multimedia Delphi components 15 years ago, Windows was the only supported platform, and all the components were built for it.

Over the last few years, Delphi rapidly expanded into more platforms, and emerged as one of the leading choices for native cross platform development. Although a challenging task, already the majority of the Multimedia, and Computer Vision components that I have developed are ported to support multiple platforms, with the goal of having practically all of them available on all platforms.
The Multimedia support in Windows also went through some evolution over the years.

When I started working with Multimedia, the WaveAPI, and the AVICodec APIs were the APIs available to Windows out of the box, and the more advanced Direct Show was available as a separated install. VideoLab was the first Multimedia component library that I developed, and its first version supported only the APIs available for Windows out of the box.

Since **VideoLab** was developed on top of **OpenWire** and it was independent of any specific API the library rapidly expanded to support Direct Show, DirectX Media Objects (DMO), the Windows Media Encoder API, and number of open source APIs such as FreeFrame, Virtual DUB, and FFMPeg/LibAV.

In the meantime I also introduced **AudioLab** for Audio processing, and it also expanded to cover multitude of technologies, including WaveAPI, Direct Show, Windows Media Encoder, DMO, ASIO, Vorbis, Speex, VST2, and VST3.

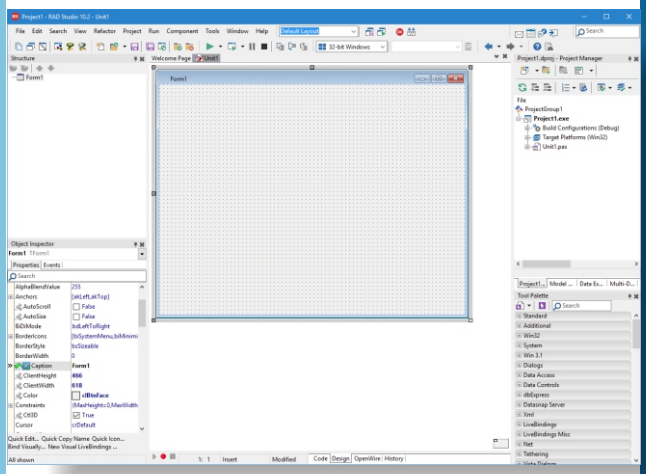
The underlying **OpenWire** architecture allows mixing the technologies in any way you want, giving the libraries unprecedented flexibility, when implementing complex Multimedia applications. The **OpenWire** architecture also allowed easily expanding the video processing functionality to add **Computer Vision** and the **VisionLab** was introduced further expanding the available functionality.

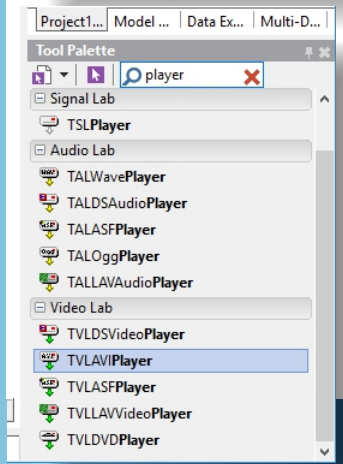
In this article, I will introduce you to the basics of the Video processing. In the following articles I will show you how to make more complex Multimedia applications, and then how to perform Computer Vision over the video.

In the first demo, we will use the venerable old school **Windows Multimedia API.** The advantage of it is that it is very lightweight, and the video streams tend to start and stop faster than when using **DirectShow.**
After this I will show you how easy it is to change the project to use the more advanced DirectShow component.

In this article I will demonstrate creating a VCL application, however creating a Fire Monkey application is identical, and many of the components are already available on MAC, iOS, Android, and even Linux.

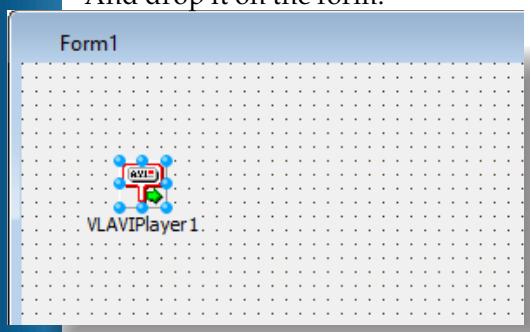
Start a new VCL Form application:



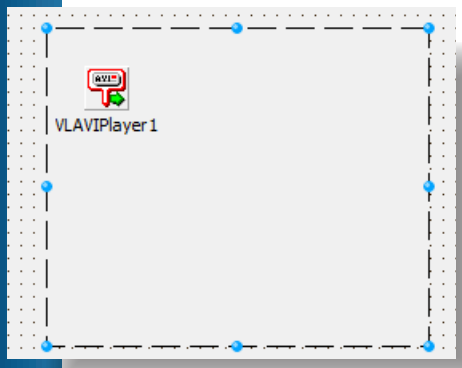


Type "player" in the **Tool Palette** search box, then select `TVLAVIPlayer` component from the palette:

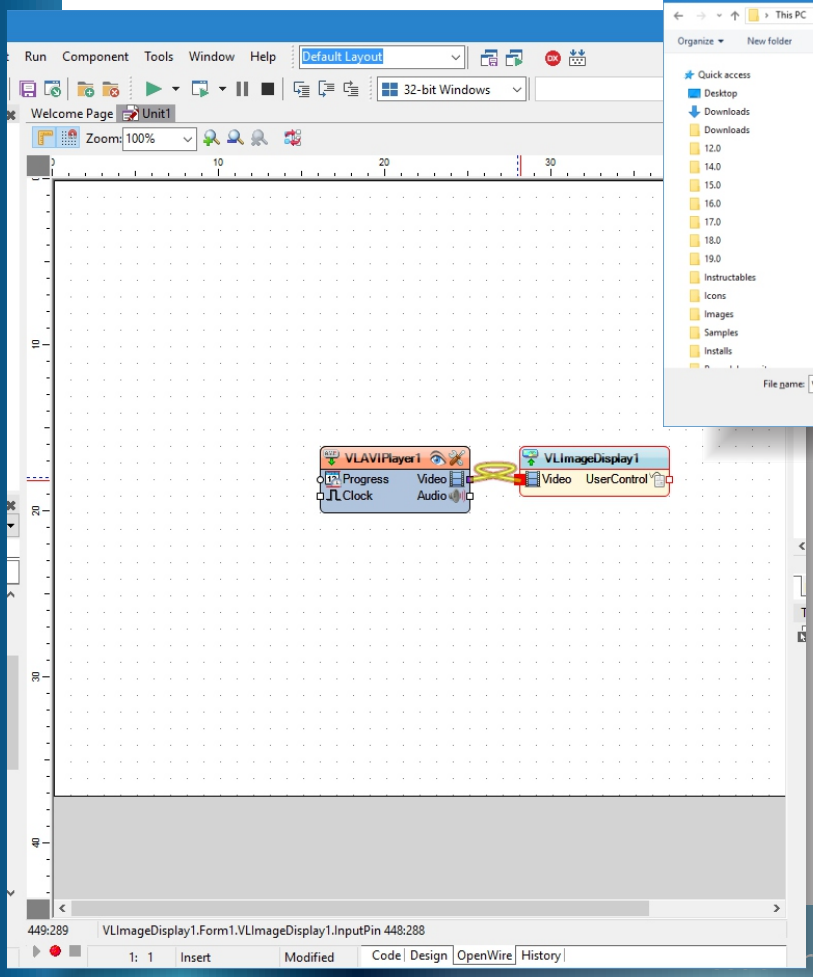
And drop it on the form:



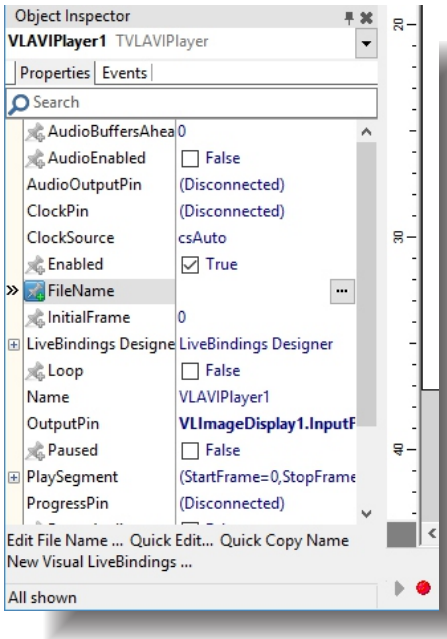
Type "display" in the Tool Palette search box, then select `TVLImageDisplay` from the palette, and drop it on the form:



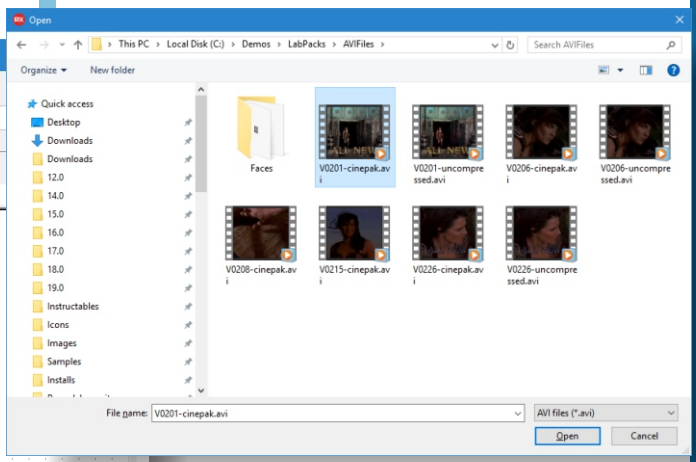
Switch to the "Open Wire" tab, and connect the "Video" Output Pin of the `VLAVIPlayer1` to the "Video" Input Pin of the `VLImageDisplay1`:



Select the `VLAVIPlayer1` component.
In the Object Inspector select the "FileName" property, and click on the ellipsis ("…") button:

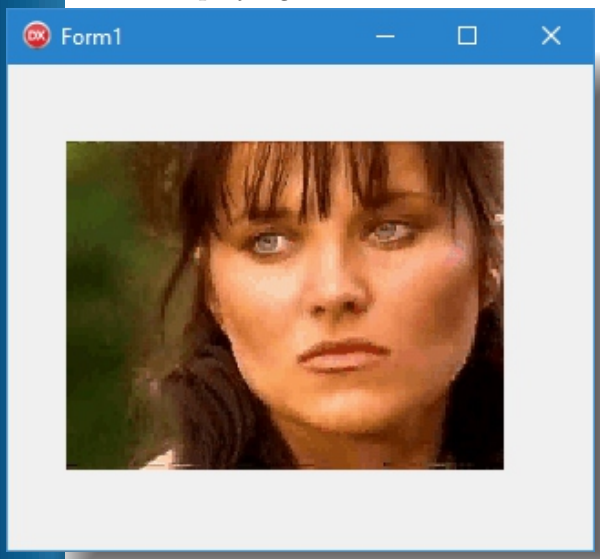


In the file dialog, select a video file to play, and click the "Open" button:

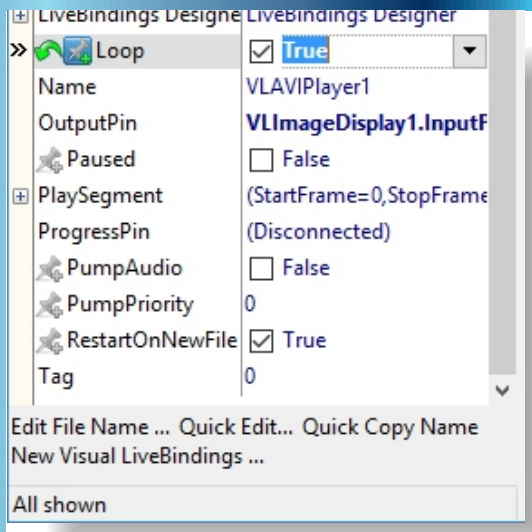


The VIDEO LAB you can get here:
`http://www.mitov.com/products/videolab#overview`
It is fully functional and you can work with it for free but you will get a "nagging information screen"
`http://www.mitov.com/products/videolab#downloads`

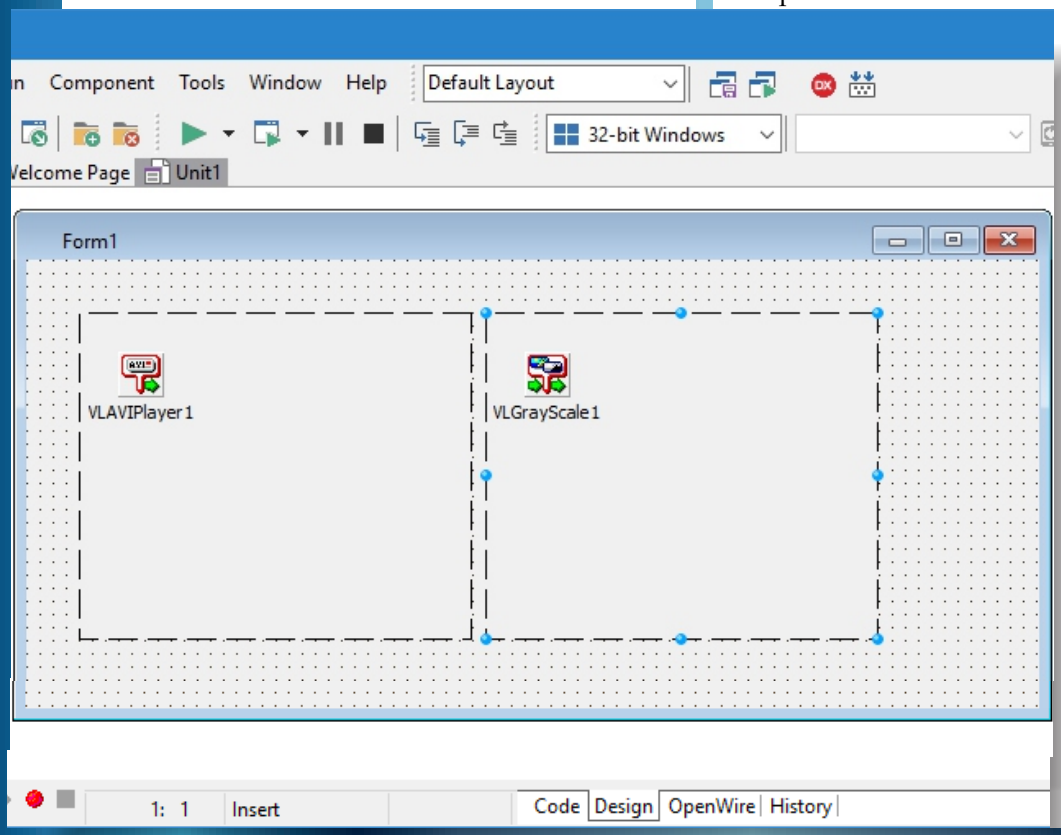
The **AVI Player** can decode only a limited number of video types, so to be sure that it will be able to decode the selected video, it is best to use one of the videos included in the **VideoLab** installation. Compile and run the application. You should see the video playing:



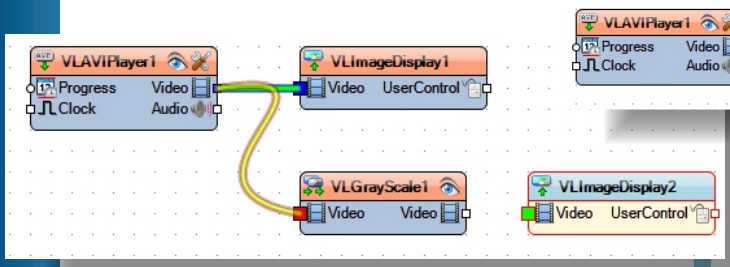
After the video finishes playing, it will stop. If you want the video to restart playing from the beginning, you can set the "Loop" property to "True" :



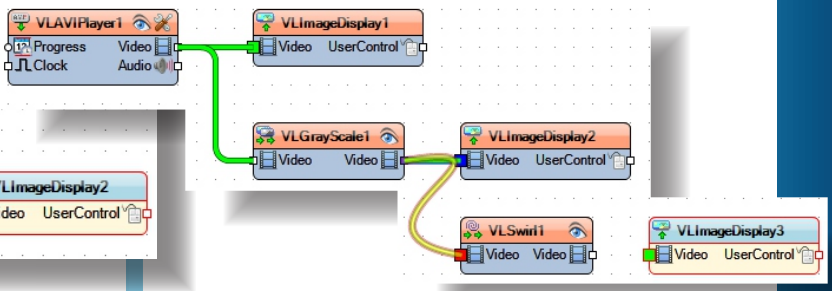
Now that you know how to play and display video, it is time to perform some image processing. One of the simplest forms of processing is to convert the video to Gray Scale. Switch to the Form Designer, and add `TVLGrayScale` and a second `TVLImageDisplay` components:



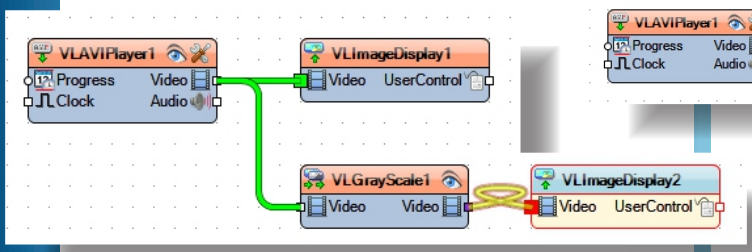
Switch to the "Open Wire" tab, and connect the "Video" Output Pin of the `VLAVIPlayer1` to the "Video" Input Pin of the `VLGrayScale1` component:



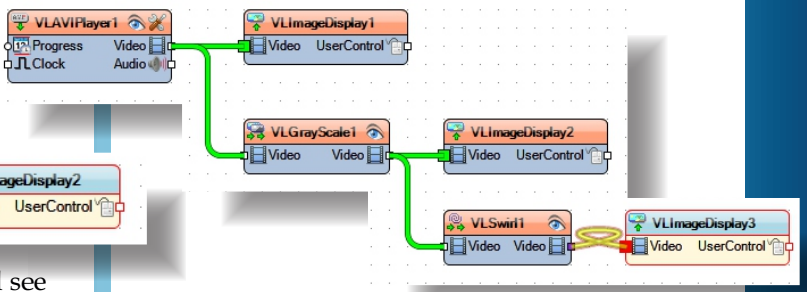
Switch to the "Open Wire" tab, and connect the "Video" Output Pin of the `VLGrayScale1` to the "Video" Input Pin of the `VLSwirl1` component:



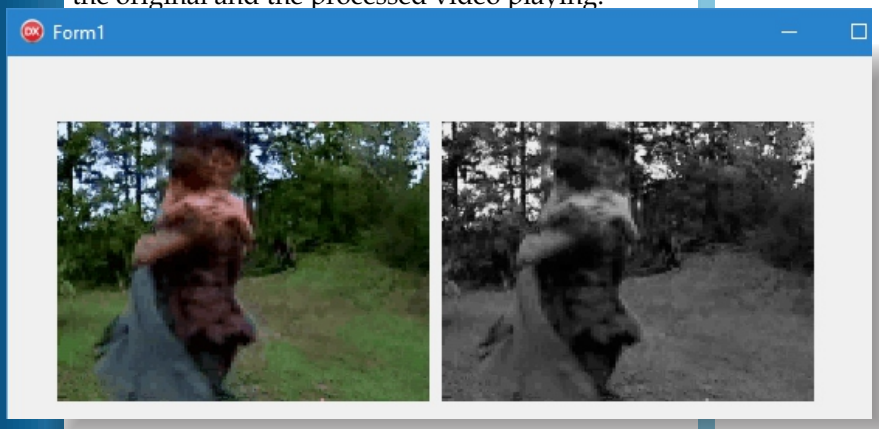
Next connect the "Video" Output Pin of the `VLGrayScale1` to the "Video" Input Pin of the `VLImageDisplay2` component:



Next connect the "Video" Output Pin of the `VLSwirl1` to the "Video" Input Pin `VLImageDisplay3` component:

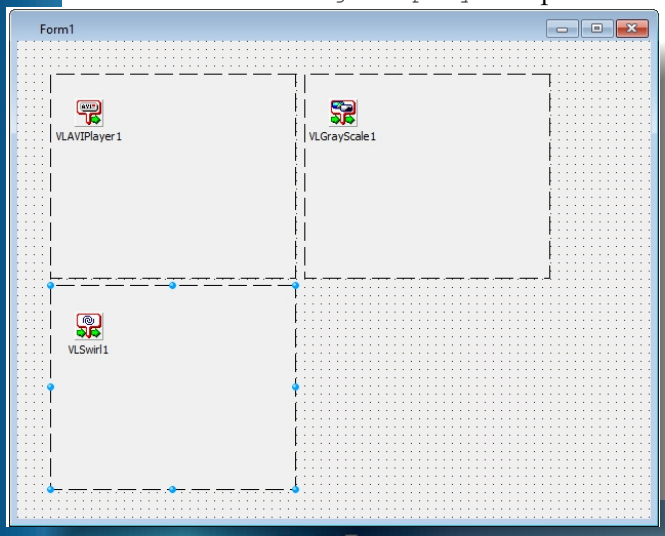


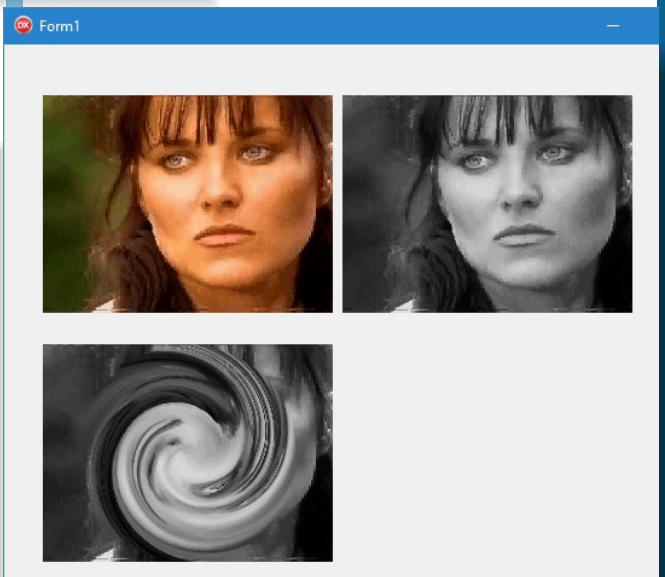
Compile and run the application. You should see the original and the processed video playing:



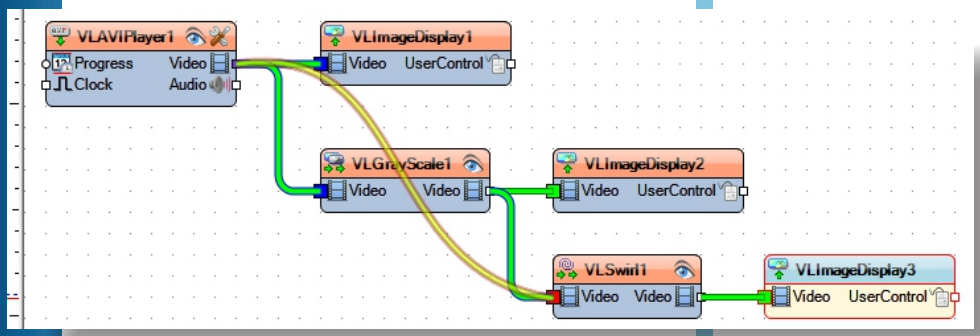
Compile and run the application. You should see the original and the processed video playing in the 3 displays:

Now we can add **Swirl effect** to the video. Switch to the **Form Designer,** and add `TVLSwirl` and another `TVLImageDisplay` components:

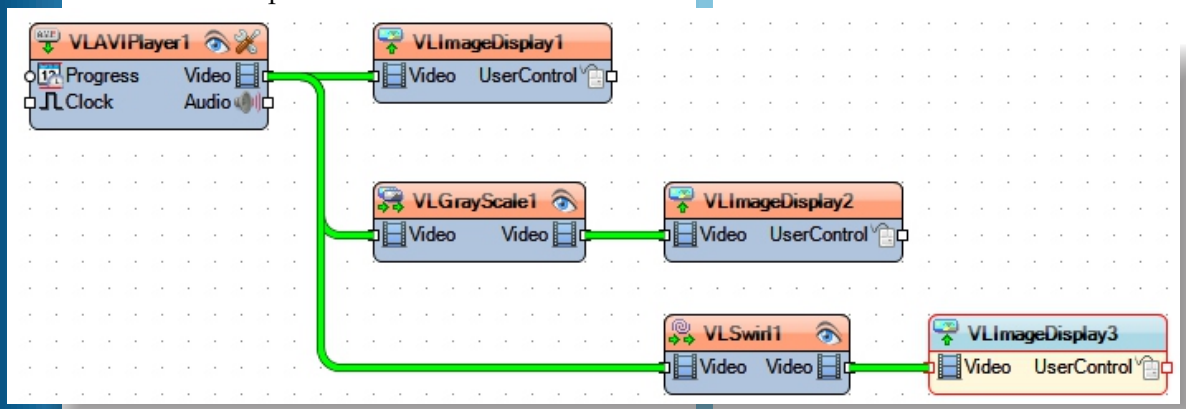




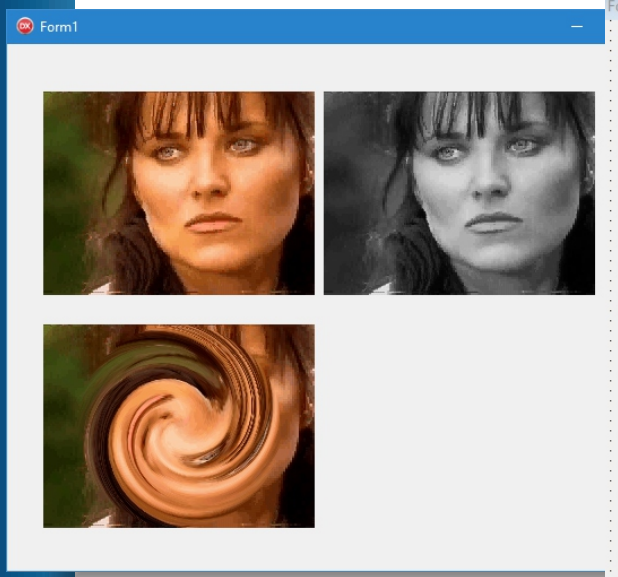
Here we perform the Swirl effect on the Gray Scale video, but we can change the project to Swirl the original video.
Switch to the "Open Wire" tab, and connect the "Video" Output Pin of the `VLAVIPlayer1` to the "Video" Input Pin of the `VLSwirl1` component:



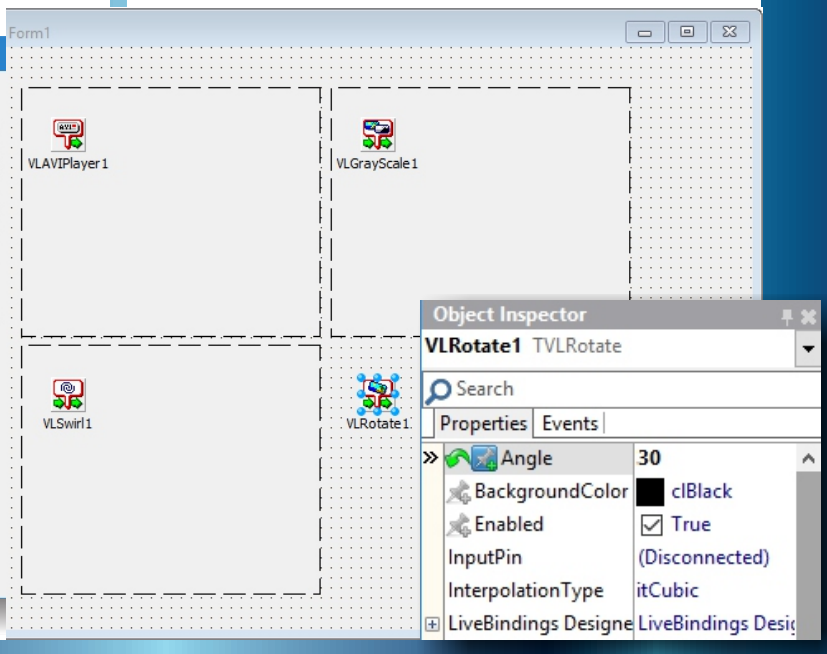
The OpenWire will automatically disconnect the connection between the `VLGrayScale1` and `VLSwirl1` components:



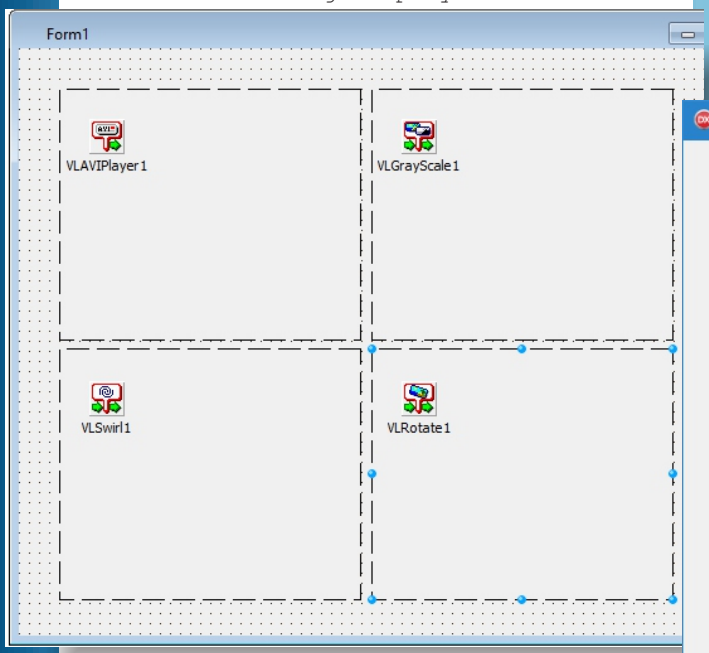
Compile and run the application. You should see the original and the processed video playing in the 3 displays, and the Swirl Display will show the Swirl of the original video:



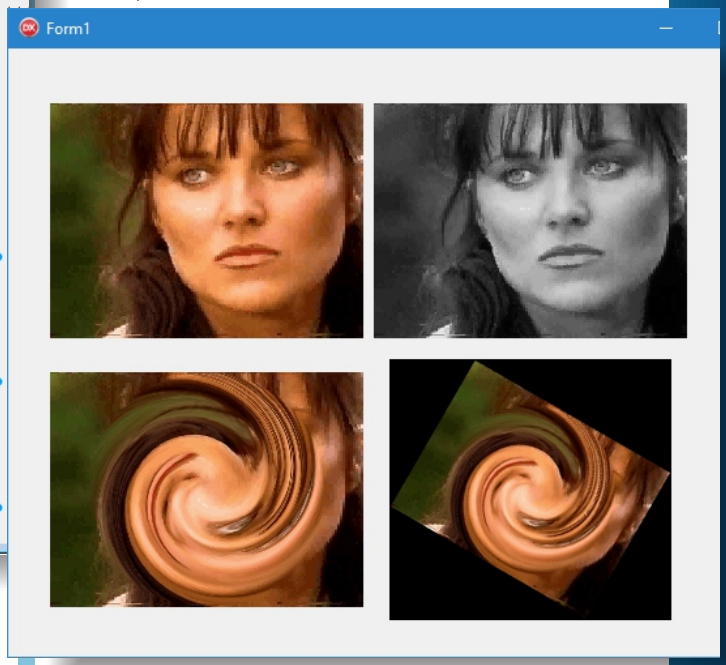
Next we will add some rotation of the swirled video. Switch to the Form Designer, and add `TRotate` component. In the Object Inspector, set the value of the "Angle" property to 30:



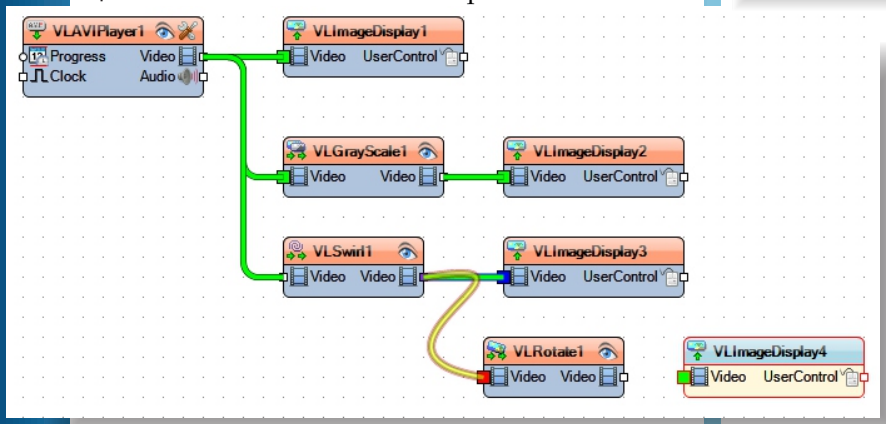
Add another `TVLImageDisplay` on the form:



Compile and run the application. You should see the original and the processed video playing in the 4 displays, and the 4th display will show the rotated, and swirled video:

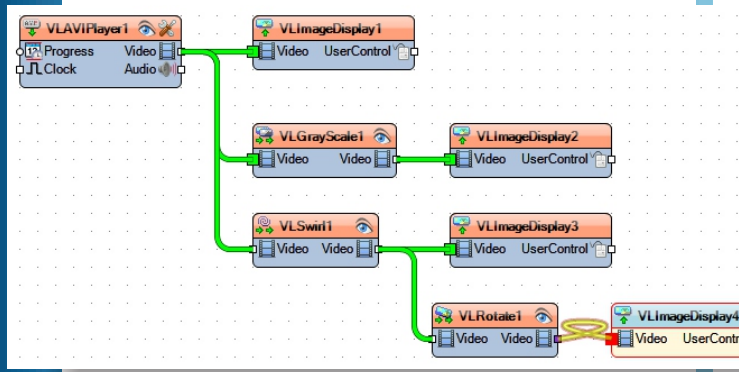


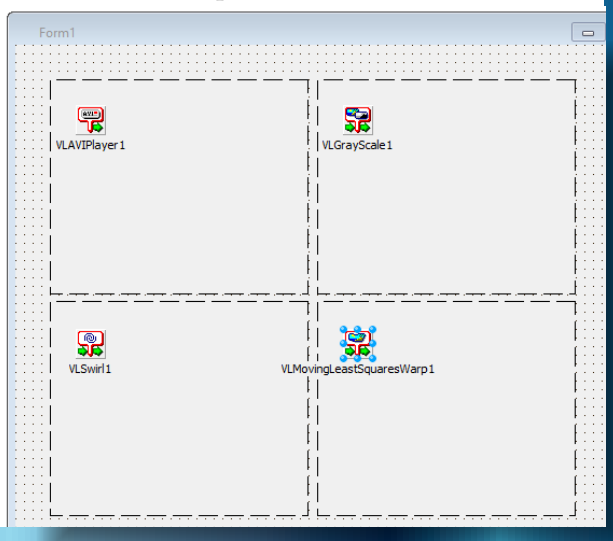
Switch to the "Open Wire" tab, and connect the "Video" Output Pin of the VLSwirl1 to the "Video" Input Pin of the `VLRotate1` component:



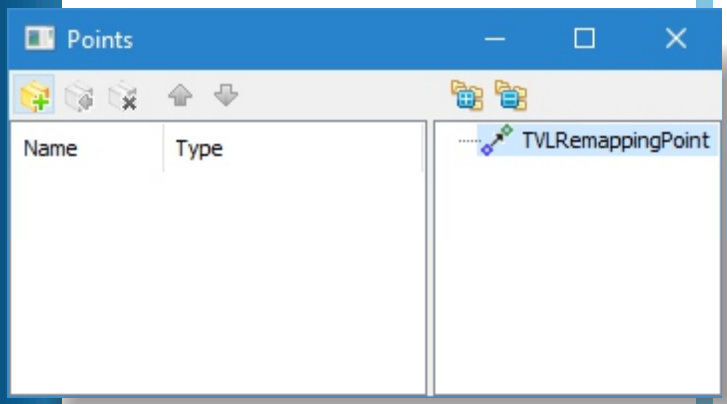
We can easily change the rotation with a more complex deformation. **VideoLab** includes a powerful remapping component called `TVLMovingLeastSquaresWarp`. It allows you to specify a number of points in the original video, and where they should be located in the processed video. This allows for custom elastic deformations of the video.
Switch to the Form Designer, and remove the `VLRotate1` component. Add a `TVLMovingLeastSquaresWarp` component on the form:

Next connect the "Video" Output Pin of the `VLRotate1` to the "Video" Input Pin of the `VLImageDisplay4` component:

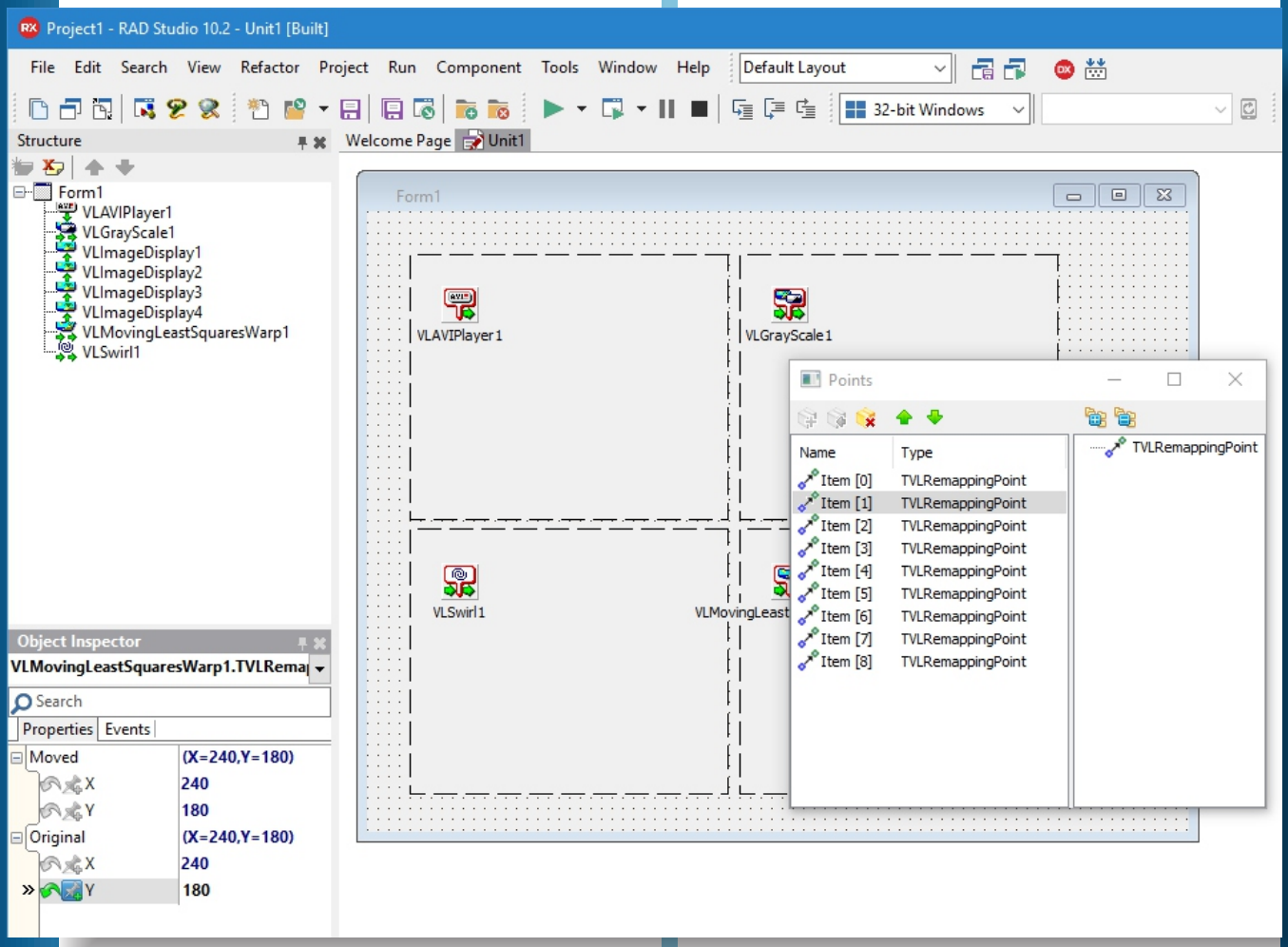




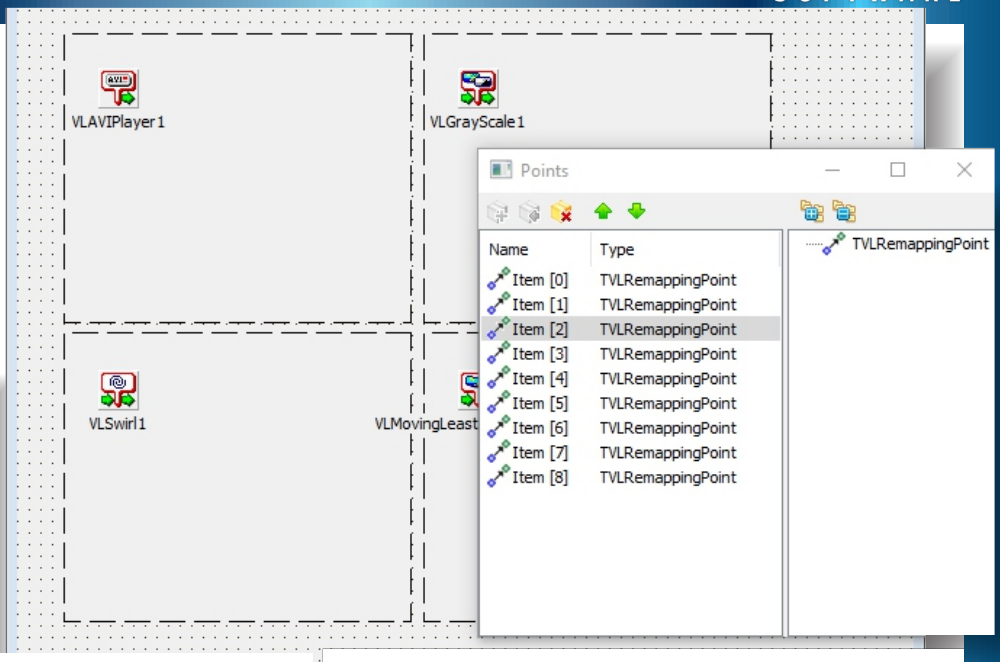
Double click on the `VLMovingLeastSquaresWarp1` component to open the points editor.
Select the `TVLRemappingPoint` on the right, and click on the "+" button on the very left 9 times to add 9 Points:



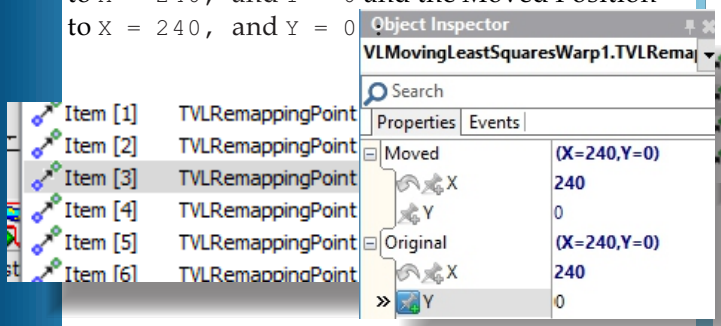
Leave the first point unchanged, and select the second point (Item [1]).
In the Object Inspector, set the Original position to `X = 240,` and `Y = 180` and the Moved Position to `X = 240,` and `Y = 180` :



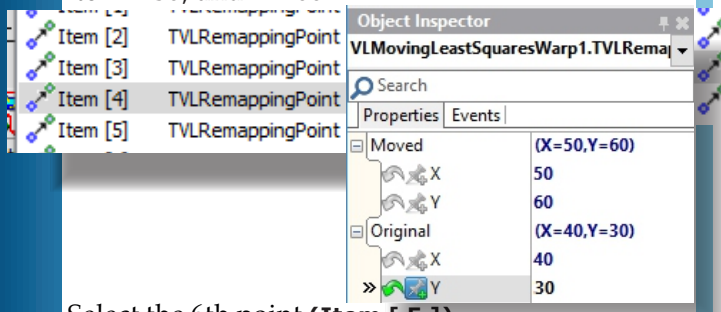
Select the third point **(Item [2])**.
In the Object Inspector, set the Original position to X = 0, and Y = 180 and the Moved Position to X = 0, and Y = 180 :



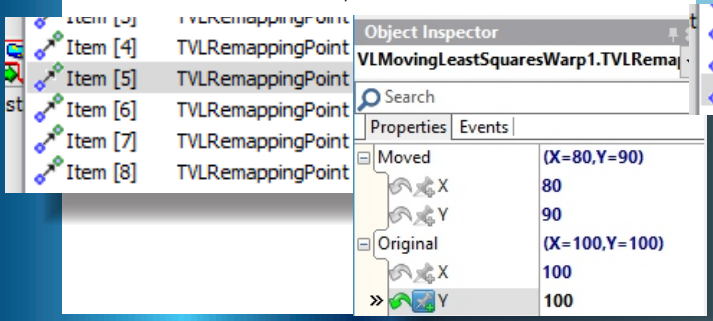
Select the 4th point **(Item [3])**.
In the Object Inspector, set the Original position to X = 240, and Y = 0 and the Moved Position to X = 240, and Y = 0



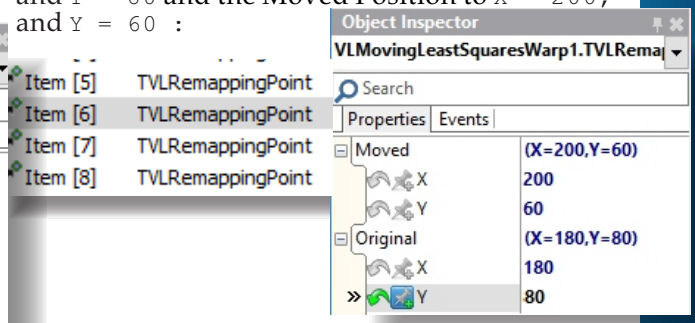
Select the 5th point **(Item [4])**.
In the Object Inspector, set the Original position to X = 40, and Y = 30 and the Moved Position to X = 50, and Y = 60



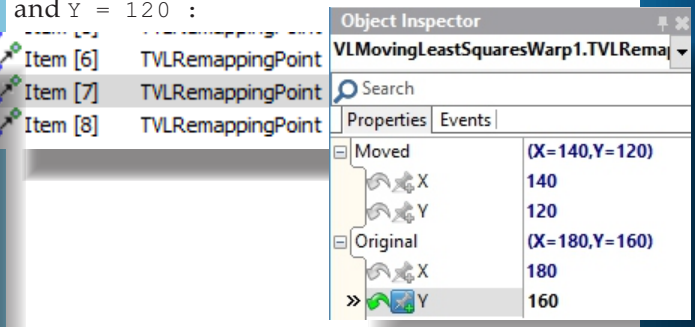
Select the 6th point **(Item [5])**.
In the Object Inspector, set the Original position to X = 100, and Y = 100 and the Moved Position to X = 80, and Y = 90 :



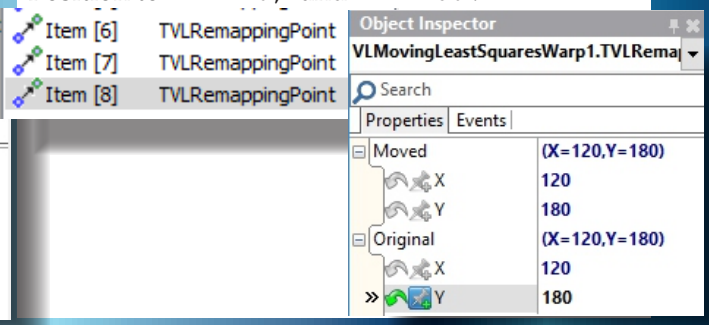
Select the 7th point **(Item [6])**. In the Object Inspector, set the Original position to X = 180, and Y = 80 and the Moved Position to X = 200, and Y = 60 :



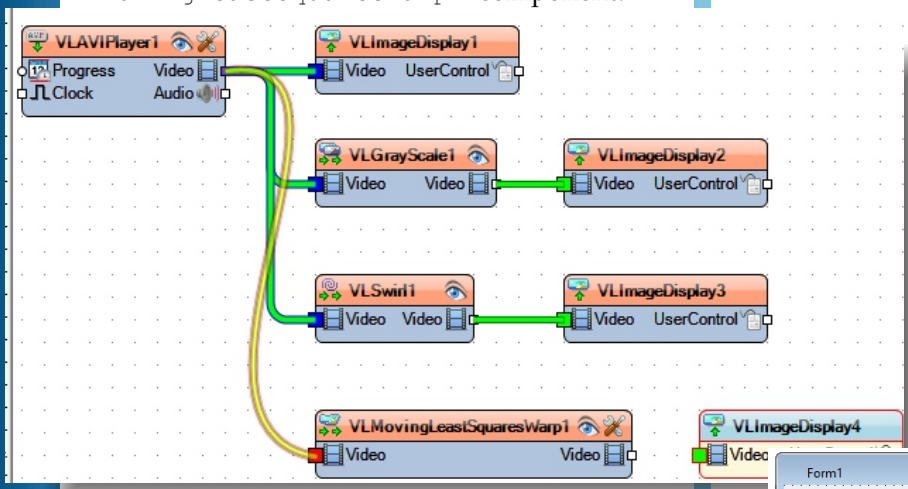
Select the 8th point **(Item [7])**. In the Object Inspector, set the Original position to X = 180, and Y = 160 and the Moved Position to X = 140, and Y = 120 :



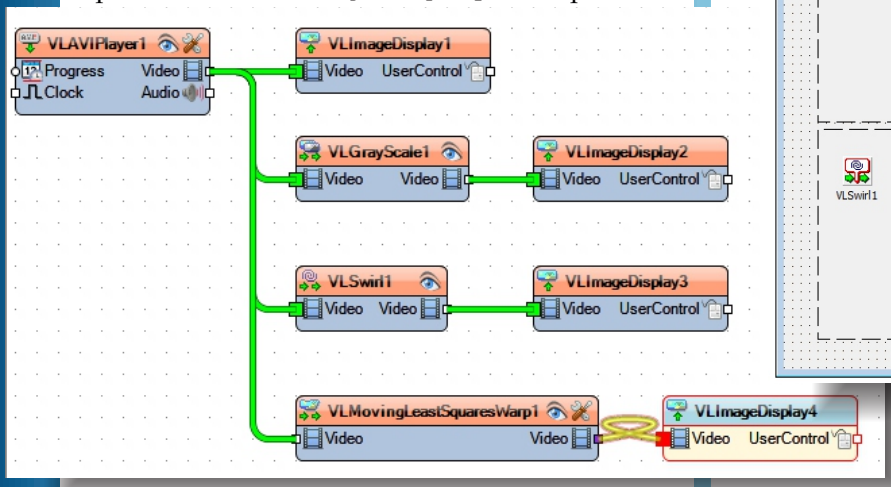
Select the 9th point **(Item [8])**.
In the Object Inspector, set the Original position to X = 120, and Y = 180 and the Moved Position to X = 120, and Y = 180 :



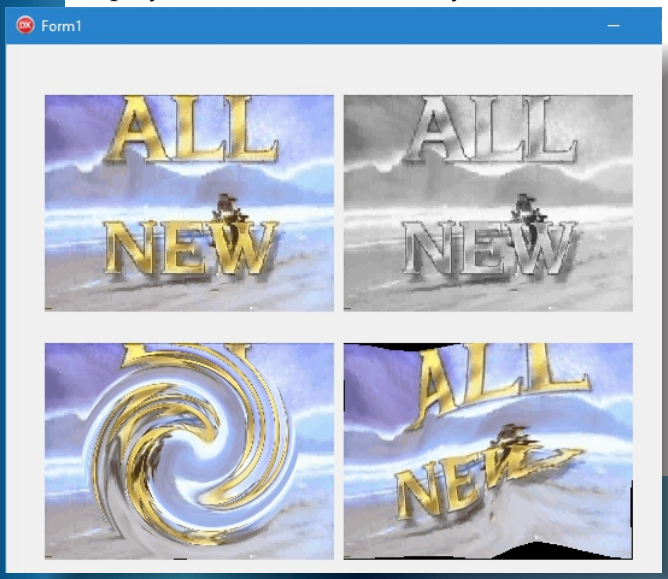
Close the Points editor.
Switch to the "Open Wire" tab, and connect the "Video" Output Pin of the `VLAVIPlayer1` to the "Video" Input Pin of the `VLMovingLeastSquaresWarp1` component:



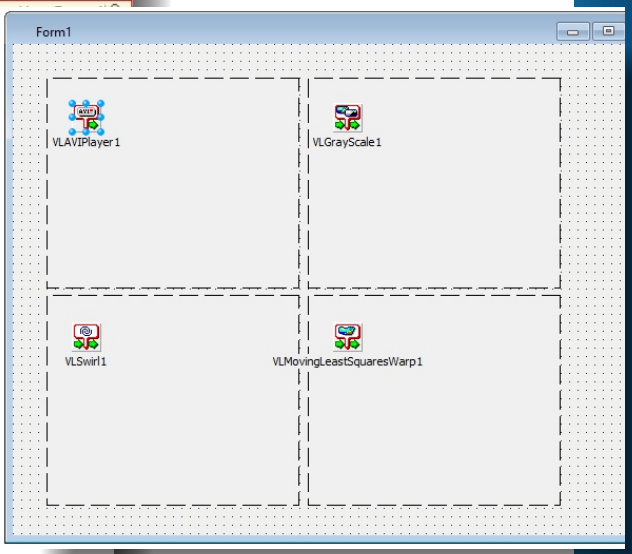
Next connect the "Video" Output Pin of the `VLMovingLeastSquaresWarp1` to the "Video" Input Pin of the `VLImageDisplay4` component:



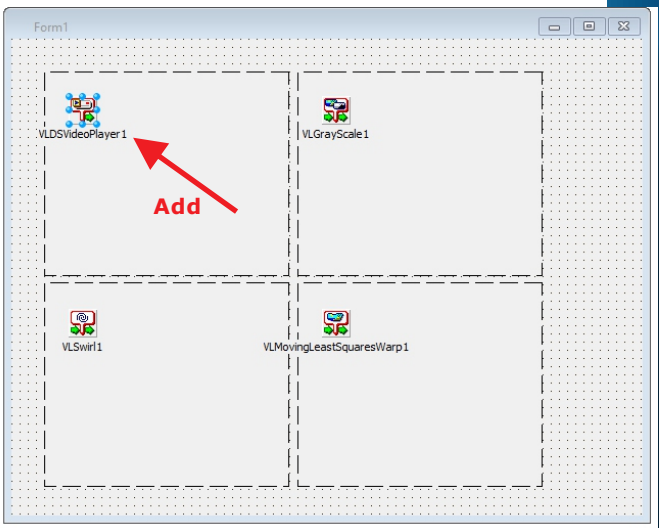
Compile and run the application.
You should see the original and the processed video playing in the 4 displays, and the 4th display will show the elastically deformed video:



Until now we used the very basic `TVLAVIPlayer` component. VideoLab has a number of other more advanced Video Player components.
It is very easy to change the project to use a different component. As example we can change the component to the `DirectShow` based `TVLDSVideoPlayer`.
Switch to the Form Designer, and select the `VLAVIPlayer1` component:

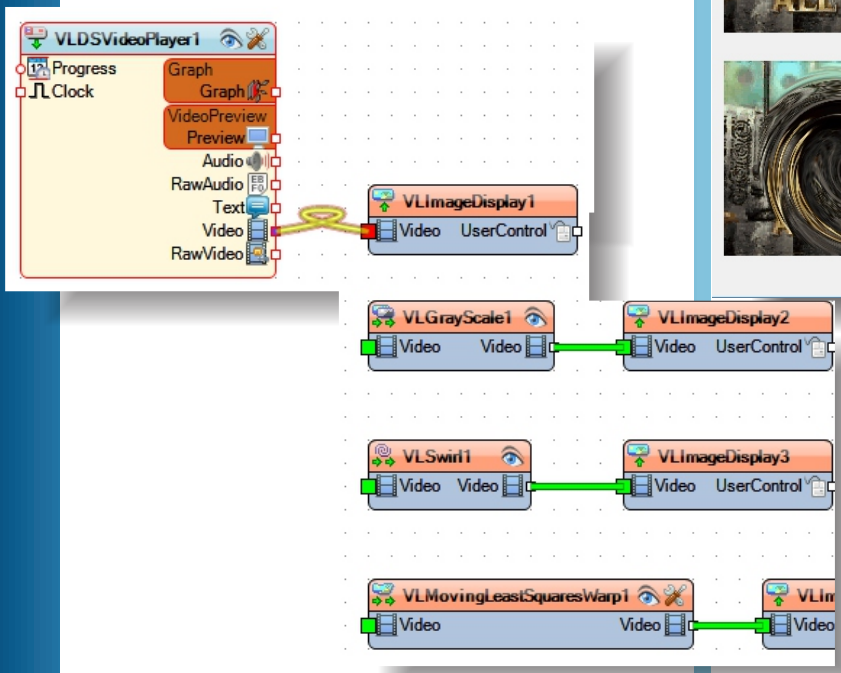


Delete the component.
Add a `TVLDSVideoPlayer` component on the form:



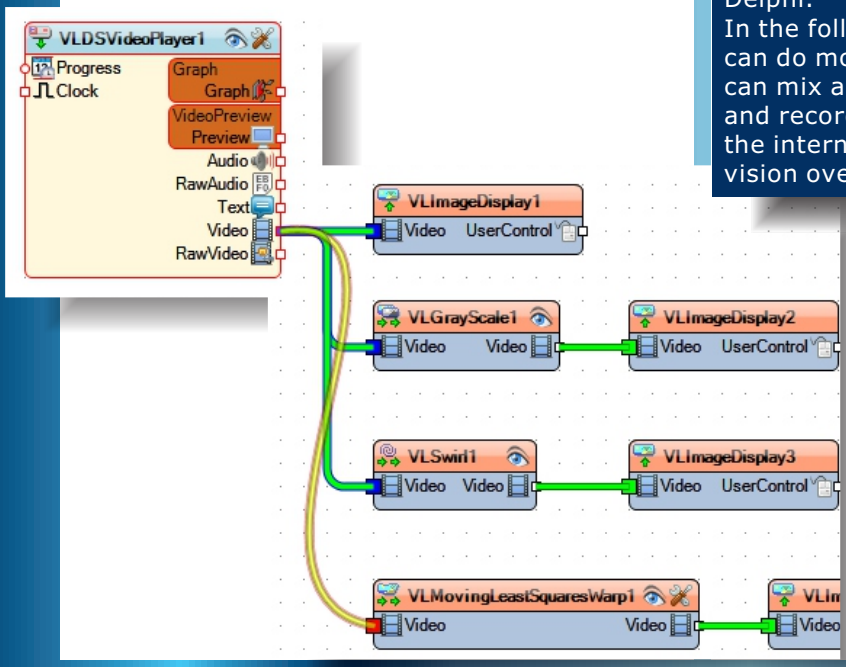
Double click on the component to open the file select dialog, and select file to play.
The `DirectShow` component can play many more file formats than the **TVLAVIPlayer.**

Switch to the "Open Wire" tab, and connect the "Video" Output Pin of the `VLDSVideoPlayer1` to the "Video" Input Pin of the `VLImageDisplay1` component:



Compile and run the application. You should see the original and the processed video playing in the 4 displays, and the 4th display will show the elastically deformed video:

Continue connecting the "Video" Output Pin of the `VLDSVideoPlayer1` to the "Video" Input Pins of the `VLGrayScale1`, `VLSwirl1` and `VLMovingLeastSquaresWarp1` components:



CONCLUSION
Following the same steps you can change the application to use any other Video Player component or video source.
In this article I showed you how easy it is to create video playing and processing applications in Delphi.
In the following articles, I will show you how you can do more complex video processing, how you can mix and analyze video, how you can capture and record video, how you can broadcast it over the internet, and how you can perform computer vision over the video.

The VIDEO LAB you can get here:
`http://www.mitov.com/products/videolab#overview`
It is fully functional and you can work with it for free but you will get a "nagging information screen"
`http://www.mitov.com/products/videolab#downloads`

FILELESS MALWARE PAGE 1/3

BY THE EDITOR

These days there is a growing division between those who believe in using antivirus software, and those who do not. Some developers say that they are well able to handle any problems an occasional virus might cause, and that having to deal with an occasional unwanted attack is worth the risk compared to the disadvantages of having active antivirus software: particularly the overall slowing of processes, and in addition to sluggishness, the possible frequent alarms from false 'recognitions'.

It is enough, they say, to be very careful in handling email and internet access. Some will develop on a machine that is not connected to the web at all, or only when strictly necessary, reserving a machine specifically for email and internet connection.
Their development machines are kept independent of the net without browsers, Skype or similar software installed on them at all. Downloads are made to a virus-protected machine where files are scanned before transfer to the development machine. Other Windows developers do not have separate isolated and connected machines but rely simply on **Windows Defender** *(free)* and nothing else.

A further option, which can be combined with either of the above, is to create a complete disk image regularly (*at least once a week*) so that in the event of an attack you are never set back by more than the time since you last made a complete disk image.
You just restore the latest image from a time before the virus was present, and maybe restore also from the last day or two's specific folder backups.
I know from experience that this works well.

You may be interested to to read about the latest developments in the virus 'market': "fileless malware", in which the malware/spyware is loaded entirely in memory so that file scanning detects nothing.

The malware developer arranges memory-only malware to load a clean executable at runtime (*something that antivirus scanners will accept such as PowerShell*) and then manipulate that process to do the nasty work for them.

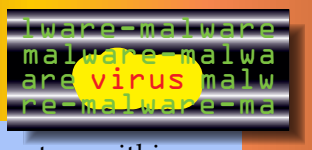
An example is a malware advertising campaign. Somewhere on the internet you encounter an ad that you glance at but browse beyond, perhaps because you have already bought the product, or already decided not to buy it. You did not click on it, but unknown to you there was an invisible iFrame which created a redirect to the malware target page of which you are unaware.

In the background the JavaScript checks (*starting from the malware landing-page*) if there are any security vulnerabilities in your browser, and if it finds any, then it sets to work. It checks whether you inside a sandbox or a virtual machine. If not, then it starts to load the full malware payload from the internet criminal into your browser's memory, which might be a 'traditional' virus, or might be fileless malware.
The full story on this can be found at
For a complete overview of this file please go to
https://www.fireeye.com/blog/threat-research/2017/03/

DISSECTING_ONE_OFAP.HTML
DISSECTING ONE OF APT29'S FILELESS WMI AND POWERSHELL BACKDOORS

At **FIREEYE** (website)is written about this subject:
Mandiant has observed APT29 using a stealthy backdoor that we call POSHSPY.

(**Mandiant** *is an American cybersecurity firm. It rose to prominence in February 2013 when it released a report directly implicating China in cyber espionage. Kevin Mandia, a former United States Air Force officer, founded Mandiant as Red Cliff Consulting in 2004 prior to rebranding in 2006. Mandiant provides incident response and general security consulting along with incident management products to major global organizations, governments, and Fortune 100 companies. Mandiant was awarded both the 2012 and 2013 SC Award for exemplary professional leadership in information-technology (IT) security. Mandiant is the creator of OpenIOC, an extensible XML schema for the description of technical characteristics that identify threats, attackers' methodologies, and evidence of compromise.*)



POSHSPY leverages two of the tools the group frequently uses: PowerShell and **"Windows Management Instrumentation"** (WMI).
In the investigations Mandiant has conducted, it appeared that **APT29** deployed POSHSPY as a secondary backdoor for use if they lost access to their primary backdoors.

POSHSPY makes the most of using built-in Windows features – so-called **"living off the land"** – to make an especially stealthy backdoor. POSHSPY's use of WMI to both store and persist the backdoor code makes it nearly invisible to anyone not familiar with the intricacies of WMI. Its use of a PowerShell payload means that only legitimate system processes are utilized and that the malicious code execution can only be identified through enhanced logging or in memory.

The backdoor's infrequent beaconing, traffic obfuscation, extensive encryption and use of geographically local, legitimate websites for command and control (C2) make identification of its network traffic difficult. Every aspect of POSHSPY is efficient and covert.

Mandiant initially identified an early variant of the POSHSPY backdoor deployed as PowerShell scripts during an incident response engagement in 2015. Later in that same engagement, the attacker updated the deployment of the backdoor to use WMI for storage and persistence. Mandiant has since identified POSHSPY in several other environments compromised by APT29 over the past two years.

WINDOWS MANAGEMENT INSTRUMENTATION

WMI is an administrative framework that is built into every version of Windows since 2000. WMI provides many administrative capabilities on local and remote systems, including querying system information, starting and stopping processes, and setting conditional triggers. WMI can be accessed using a variety of tools, including the Windows WMI Command-line (wmic.exe), or through APIs accessible to programming and scripting languages such as PowerShell.
Windows system WMI data is stored in the WMI common information model (CIM) repository, which consists of several files in the `System32\wbem\Repository` directory.

WMI classes are the primary structure within WMI. WMI classes can contain methods (code) and properties (data). Users with sufficient system-level privileges can define custom classes or extend the functionality of the many default classes.

WMI permanent event subscriptions can be used to trigger actions when specified conditions are met. Attackers often use this functionality to persist the execution of backdoors at system start up. Subscriptions consist of three core WMI classes: a Filter, a Consumer, and a FilterToConsumerBinding. WMI Consumers specify an action to be performed, including executing a command, running a script, adding an entry to a log, or sending an email. WMI Filters define conditions that will trigger a Consumer, including system startup, the execution of a program, the passing of a specified time and many others.

A FilterToConsumerBinding associates Consumers to Filters. Creating a **WMI** permanent event subscription requires administrative privileges on a system. We have observed **APT29** use WMI to persist a backdoor and also store the **PowerShell** backdoor code. To store the code, **APT29** created a new WMI class and added a text property to it in order to store a string value. **APT29** wrote the encrypted and **base64-encoded PowerShell backdoor** code into that property.

APT29 then created a **WMI** event subscription in order to execute the backdoor. The subscription was configured to run a PowerShell command that read, decrypted, and executed the backdoor code directly from the new WMI property.
This allowed them to install a persistent backdoor without leaving any artifacts on the system's hard drive, outside of the WMI repository. This **"fileless"** backdoor methodology made the identification of the backdoor much more difficult using standard host analysis techniques.

POSHSPY is an excellent example of the skill and craftiness of APT29. By "living off the land" they were able to make an extremely discrete backdoor that they can deploy alongside their more conventional and noisier backdoor families, in order to help ensure persistence even after remediation. As stealthy as POSHSPY can be, it comes to light quickly if you know where to look. Enabling and monitoring enhanced PowerShell logging.

APT29 then created a WMI event subscription in order to execute the backdoor. The subscription was configured to run a PowerShell command that read, decrypted, and executed the backdoor code directly from the new WMI property. This allowed them to install a persistent backdoor without leaving any artifacts on the system's hard drive, outside of the WMI repository. This "fileless" backdoor methodology made the identification of the backdoor much more difficult using standard host analysis techniques.

POSHSPY is an excellent example of the skill and craftiness of APT29.
By "living off the land" they were able to make an extremely discrete backdoor that they can deploy alongside their more conventional and noisier backdoor families, in order to help ensure persistence even after remediation.

As stealthy as POSHSPY can be, it comes to light quickly if you know where to look.
Enabling and monitoring enhanced PowerShell logging **(https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html)** can capture malicious code as it executes and legitimate WMI persistence is so rare that malicious persistence quickly stands out when enumerating it across an environment.
This is one of several sneaky backdoor families that we have identified, including an off-the-shelf domain fronting backdoor **https://www.fireeye.com/blog/threat-research/2017/03/apt29_domain_frontin.html**

and **HAMMERTOSS.**

(https://www.fireeye.com/blog/threat-research/2015/07/hammertoss_stealthy.html)
When responding to an APT29 breach, it is vital to increase visibility, fully scope the incident before responding and thoroughly analyze accessed systems that don't contain known malware.

ADDITIONAL READING
This PowerShell logging blog post

https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html

contains more information on improving PowerShell visibility in your environment.
This excellent whitepaper by William Ballenthin, Matt Graeber and Claudiu Teodorescu contains additional information on WMI offense, defense and forensics.

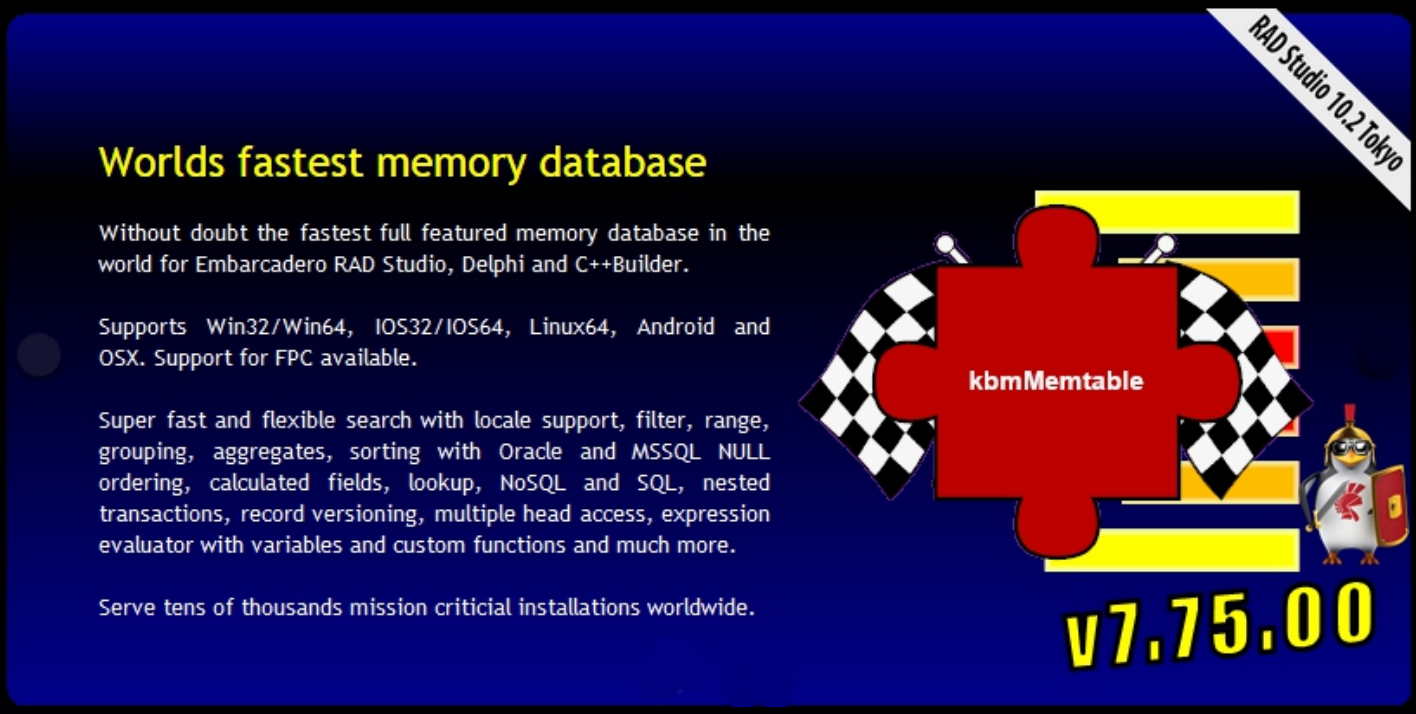
https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-windows-management-instrumentation.pdf

This presentation by Christopher Glyer and Devon Kerr contains additional information on attacker use of WMI in past Mandiant investigations.

https://files.sans.org/summit/Digital_Forensics_and_Incident_Response_Summit_2015/PDFs/TheresSomethingAboutWMIDevonKerr.pdf

The **FireEye FLARE** team released a WMI repository-parsing tool that allows investigators to extract embedded data from the WMI repository and identify WMI persistence.

https://github.com/fireeye/flare-wmi/tree/master/python-cim



KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.01.00 RELEASED!

- **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 4 DEVELOPERS

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.