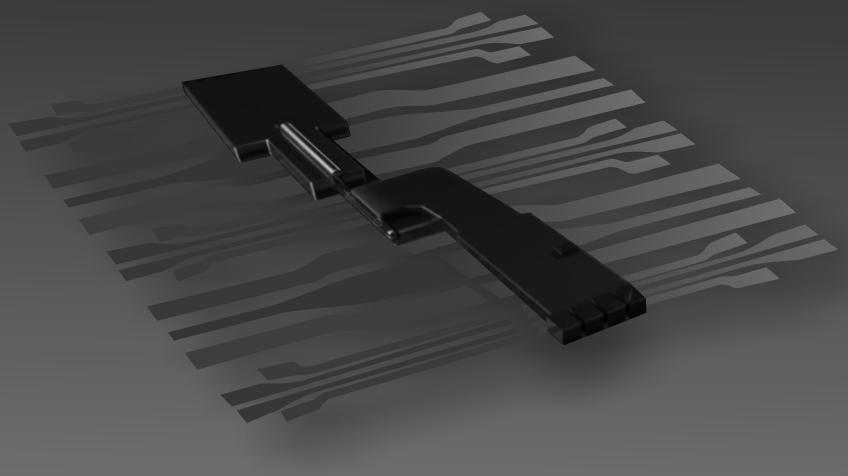
FOR DELPHI, LAZARUS, AND PASCAL RELATED LANGUAGES / ANDROID, IOS, MAC, WINDOWS & LINUX PRINTED, PDF, & ONLINE VIEW

BLAISE PASCAL MAGAZINE 66



POCKET EDITION

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

LIBRARY STICK

All issues on the USB stick complete searchable 3360 pagesfully indexed

GET THE BOOK INCLUDING THE NEWEST LIBRARY STICK INCLUDING 1 YEAR DOWNLOAD

COMBINATION: 3 FOR 1

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

75€

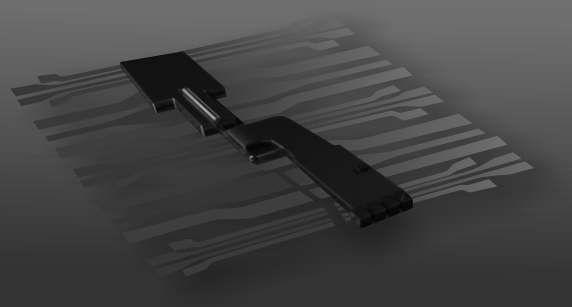
https://www.blaisepascal.eu/subscribers/UK/UK_CD_DVD_USB_Department.html

BLAISE PASCAL MAGAZINE 66

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

CONTENTS

ARTICLES



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, Peter Bijlsma

Subscriptions (2017 prices)

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

Member and donator of

Printed magazine edition
10 issues per annum, 44-page Delphi-only section: € 90.-- This includes postage, VAT at 6 % and all code and programs accompanying the articles.
Excluding postage the 44-page edition is € 60.-- per annum.
10 issues per annum 80-page Delphi + Lazarus sections: € 150 plus € 80 for postage.
Digital magazine edition (PDF format)
10 issues per annum 80-page Delphi + Lazarus sections: € 50.-- (excluding VAt at 21%).
For the months to the end of 2017 we are trialling a fuller 80-page edition of the magazine, with two sections, the first with a Delphi focus, and the second with a Lazarus/FPC focus.
We will decide, based on reader feedback, at the end of 2017 whether to continue with this larger format magazine, or revert to the 44-page format you know from recent issues.
Subscriptions can be taken out online at www.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

Almost Christmas.
Time flew this year: I had a little trouble with my health and I still try to recover. That is why I am so late with this issue, I apologize for that. That puts double pressure on getting it all done.
There also was a spectacular development on the side of Lazarus: The Pas2Js Transpiler. It is still in Beta, but now you can use it. In the next item I will explain about how to use it and get it. The Lazarus environment will come into a new phase during the springtime next year. Lazarus is at the level of Delphi 7 and very much available for the web.
There will be quite some surprises, some very interesting stories about that and it will be explained in the next Issue.
In this issue I wrote an extra article about the quantum computer: Majorana's. About what it is and finally means for the future. I hope you will appreciate that and I will send you some questions about what you would like in the Magazine, what you think of its appeal etc.
Here is an address of something you should carefully read:
https://www.symmetrymagazine.org/article/august-2015/testing-the-nature-of-neutrinos
Some of the importance:
"This type of experiment, taking place with different types of atoms around the world, is the only kind currently able to determine whether neutrinos are Majorana particles. If they are, they could have caused an asymmetry early in the universe's history that led to the world we see today, allowing matter to win out over antimatter".
A nice Christmas message…

One other thing I had planned for, is writing an article about Bitcoins and Block chains.
As always the articles grew and new things happen so the urgency of the subject increases to be more and better explained. You will see that in the next issue…

And then there is something new happening:
Eight years ago when I was in Paris for a conference, I talked to "John Thomas", a former Inprise-official whom for the second time got in to the Delphi Business now called Embarcadero.
We then talked about the future of Delphi.
I asked why there was no interest for the internet.
It was a minor subject he said.
I was quite annoyed about that.
It seems that with Idera a new chapter starts.
I hear a lot of rumours about the Internet and new ways of making it available for Pascal.
I sincerely think that will be a good thing for Delphi!
I'll write about that in the next issue at the end of the month. It will be an enormous surprise for all of you, but yet you must wait a little...

For now I wish you all a very nice Christmas and a very peaceful time...

A mile below the Earth's surface, a large copper cylinder sits behind a thick shield of lead bricks stacked into what could be confused for a wood-burning pizza oven

Matt Kapust, Sanford Lab

BY DETLEF OVERBEEK

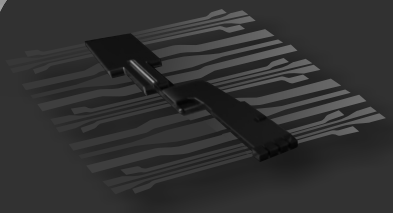


Figure: 1 Part of a quantum chip Figure: 2 Majorana Prize

MAJORANA

In the earlier article (Issue 62, 63) I described some details about the workings of the Quantum Computer. The quantum computer in short consists of two parts: the quantum part and a regular computer part.

Now the quantum part has the problem of Qubits in numbers. It has not yet been possible to create a chip which contains large numbers of qubits. Because there is still a problem of the numbers of qubits that could be placed on the Nano Chip research is still being done and now have great results.

To bring back into your remembrance the number of qubits is decisive for the number of calculations that the quantum computer can do. Majorana's seem to have the capability of putting us into the next phase of the development for quantum computers. Here is an explanation of how a Majorana works.

Figure: 3 **Prof.dr.ir. Leo Kouwenhoven** and his team in the lab. **Photo:** Sam Rentmeester.

For the first time, scientists in Leo Kouwenhoven's research group managed to create a nanoscale electronic device in which a pair of Majorana fermions 'appear' at either end of a nanowire. They did this by combining an extremely small nanowire, made by colleagues from Eindhoven University of Technology, with a superconducting material and a strong magnetic field. 'The measurements of the particle at the ends of the nanowire cannot otherwise be explained than through the presence of a pair of Majorana fermions', says Leo Kouwenhoven.

ETTORE MAJORANA

born on 5 August 1906 – probably died after 1959 was an Italian theoretical physicist who worked on neutrino masses. On March 25, 1938, he disappeared under mysterious circumstances while going by ship from Palermo to Naples. The Majorana equation and Majorana fermions are named after him. In 2006, the Majorana Prize was established in his memory.

Figure: 4 Ettore Majorana

NANOWIRES

Semiconductor nanowires are ideal for realizing various low-dimensional quantum devices.

In general, the operation of exchanging two identical particles may cause a global phase shift but cannot affect observables, can emerge when a *(semiconductor)* nanowire is brought into contact with a superconductor. To exploit the potential of quantum computing fully, they need to be exchanged in a well-**controlled braiding operation** which is shown and explained in the figures 5 and 6 and description on page 9

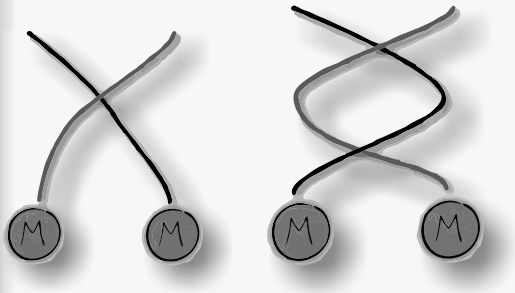


Figure: 5 and 6 controlled braiding operation

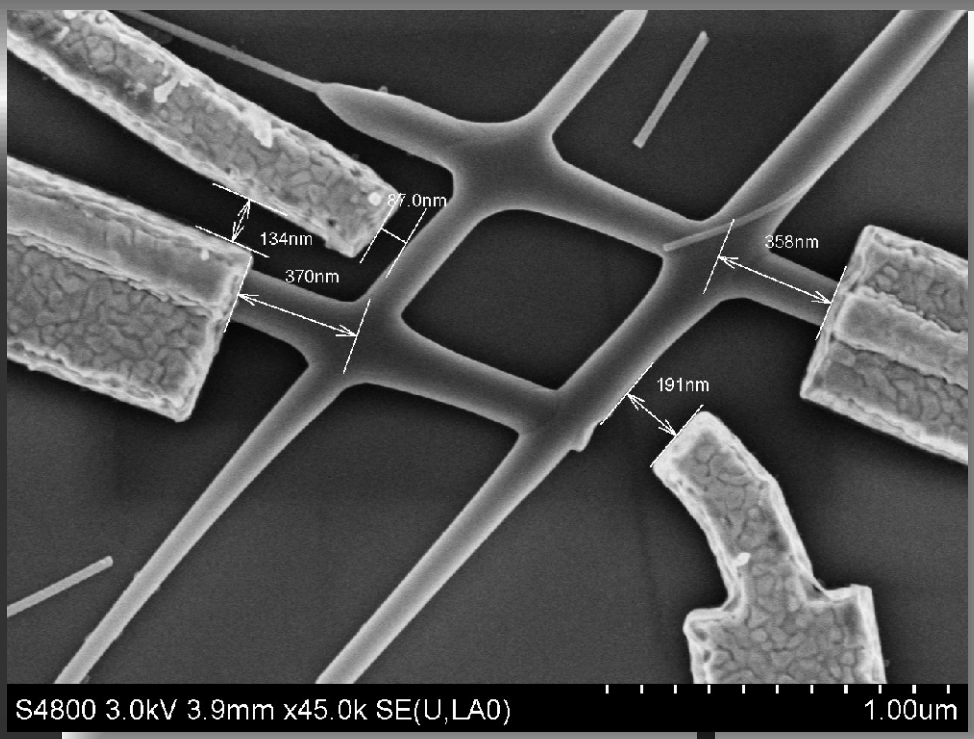


Figure: 7 The nano Hashtag

Essential hardware for **braiding** is a **network of crystalline nanowires** coupled to superconducting islands. See Figure 8...28
Here we demonstrate a technique for generic bottom-up synthesis of complex quantum devices with a special focus on nanowire networks with a predefined number of superconducting islands.

Structural analysis confirms the high crystalline quality of the nanowire junctions, as well as an epitaxial superconductor –semiconductor interface.
(*Epitaxy refers to the deposition of a crystalline overlayer on a crystalline substrate. The overlayer is called an epitaxial film or epitaxial layer.*)

Quantum transport measurements of nanowire 'hashtags' reveal a phase-coherent system is demonstrated in these hybrid nanowires, highlighting the successful materials development necessary for a first braiding experiment.
This approach of the team opens up new avenues for the realization of epitaxial three-dimensional quantum architectures which have the potential to become key components of various quantum devices.

An international team of researchers from **Eindhoven University of Technology, Delft University of Technology and the University of California – Santa Barbara** presents an advanced quantum chip that will be able to provide definitive proof of the mysterious Majorana particles.

The chip, which comprises ultrathin networks of nanowires in the shape of 'hashtags', has all the qualities to allow **Majorana** particles to exchange places. This feature is regarded as the smoking gun for proving their existence and is a crucial step towards their use as a building block for future quantum computers.

The team has built a nano-net to catch one of the most elusive particles and physics of the **Majorana particle**. **Majorana fermions** can be used for making quantum bits or qubits, the basic building block of a quantum computer which can solve specific calculations very quickly

THE MAJORANA WAS PREDICTED TO EXIST IN 1937 BY THE ITALIAN THEORETICAL PHYSICIST ETTORE MAJORANA.

There is no Majorana among the elementary particles like electrons and quarks, but in 2010 it was predicted that they could be created as **quasi particles.** A quasi particle is a collective phenomenon that behaves like a particle, a bit like a wave on the water. The water itself stays in the same place but the (*Figures 8, 9,107*) wave can travel bounce and reflect as if it were a particle.

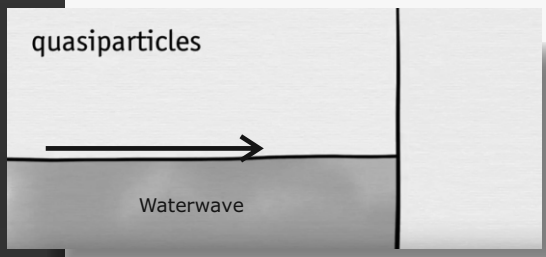


Figure: 8 Waterwave building

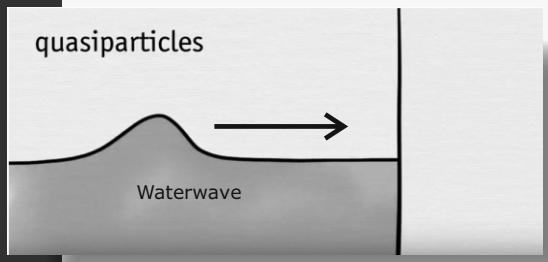


Figure: 9 The wave moving

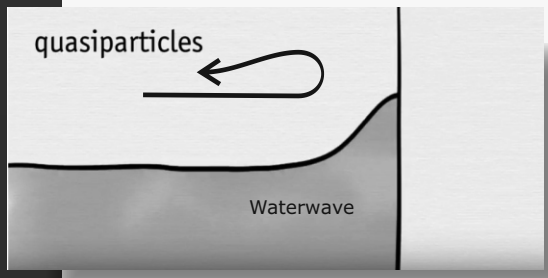


Figure: 10 The wave moves the water is still in place

HOW TO CREATE MAJORANA QUASI PARTICLES?

Start out with ultra-thin nano wires, made out of indium antimonide (*a semiconductor*), then have the nano wires border on the superconductor a material that has zero electrical resistance.

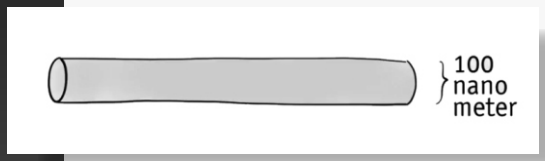


Figure: 11 The nano wires border

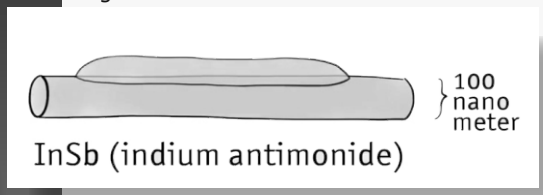


Figure: 12 Add a magnetic field.

Now the neighbouring **indium antimonide** turns into a so-called **topological superconductor** when all conditions are tuned carefully. A **Majorana quasi particle** can form at the edges of this region.

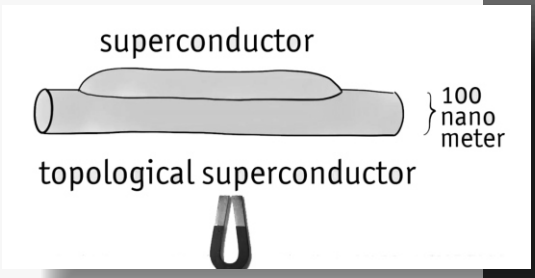


Figure: 13
The neighbouring indium antimonide turns into a toplogical superconductor

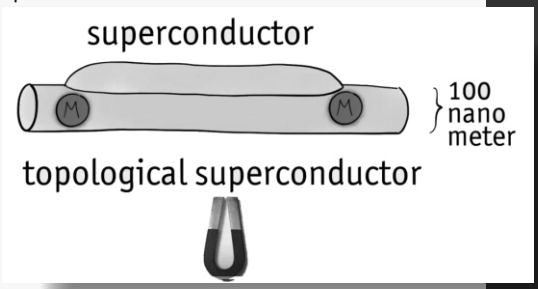


Figure: 14
A Majorana quasi particle can form at the edges of this region.

Figure: 15

In 2012 the research team including researchers at **Eindhoven and Delft University of Technology (Netherlands)** published the first detection of a **Majorana.** In this way nano wires can be grown under a tiny **gold drop** on an (*indium phosphide*) crystal.

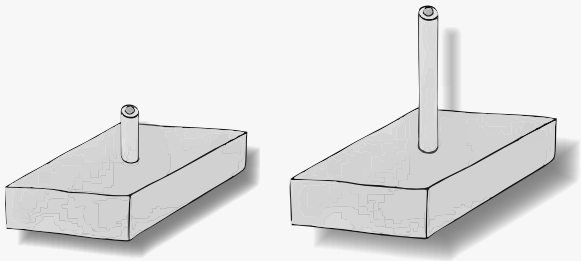
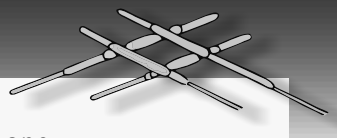


Figure: 15,16 a tiny gold drop on an crystal.



The team used a crystal with carefully crafted trenches so that the nano wires would **cross each other** on the **indium phosphide stalks**.
They grew thicker indium **antimonide nanowires.**
Then they gently sprayed the resulting hash tag shapes  with aluminium – which is a superconductor.

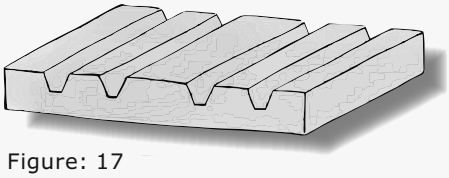


Figure: 17
crystal with carefully crafted trenches

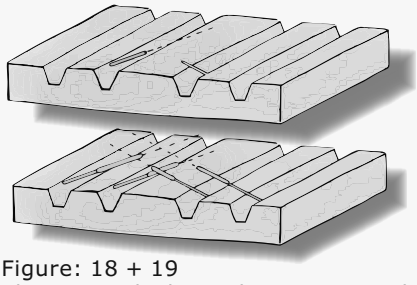


Figure: 18 + 19
They grew thicker indium antimonide nanowires.

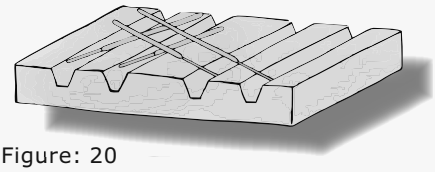


Figure: 20
The hashtag nanowires appear.

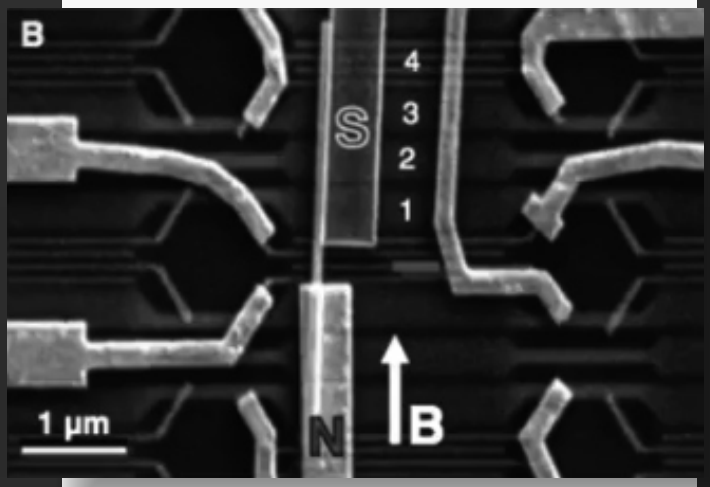


Figure: 21 Mourik, V. et al. Signatures of Majorana Fermions in Hybrid Superconductor-Semiconductor Nanowire Devices, **Science** 336, 1003 (2012)

Spraying the front wire casts a shadow on the back one.
So the coating ends right at the crossings.
This is where the maginot (*front line*) can appear (figure 22,23,24)
So each hashtag shape can produce four (4) major honours. That comes in handy for the quantum computing bit.

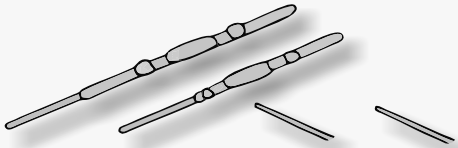


Figure: 22 Notice four rounded points

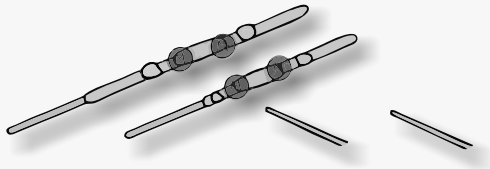


Figure: 23 This shows where the Majoranas are wanted

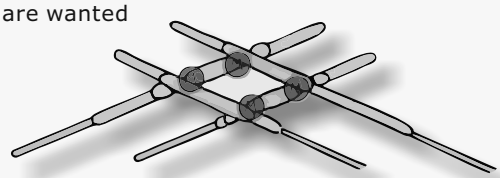


Figure: 24 The definite creation of the Hash tag

The big **selling point of Majorana particles** for quantum computers is that when you exchange two Majorana particles and exchange them again you will not be back where you started.

Quantum mechanics tells us that the double exchange is **encoded in the two Majorana** particles.

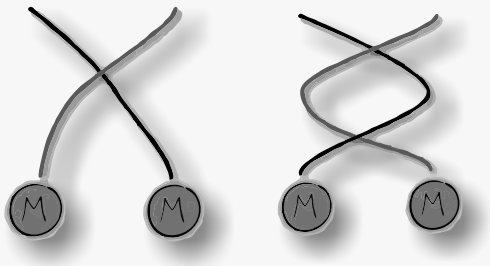


Figure: 25,26
The quantum braiding doubly exchanged

It's a bit like the way that two loose ends of string when doubly exchanged are abraded indeed doubly exchanged. (figure 25,26).
Majorana's are said to be braided.
This quantum-braiding is predicted to be very stable, as it is not easy to undo a braid without cutting.
This is a major advantage.

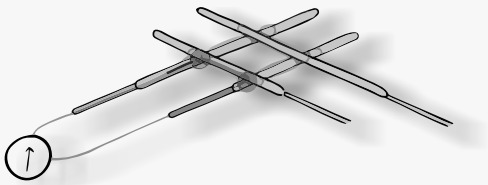


Figure: 28 A small current is added

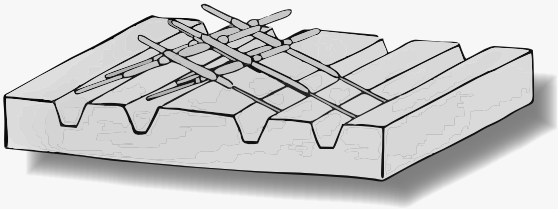


Figure: 29 The first hashtags are visisble

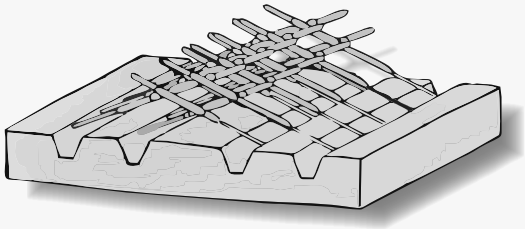


Figure: 30

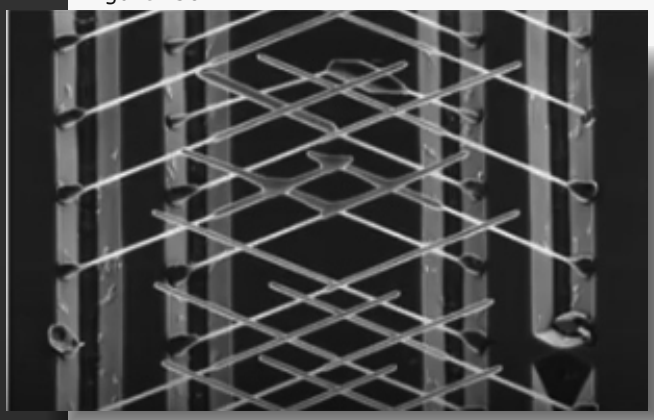


Figure: 30, 31 Schematic view and a photo graphic image

Instability or de-coherence is a major problem for most other qubit-candidates that are being researched. We have produced networks of hash tags and we have shown that these are clean enough for a readout of two major honours by **applying a small current**. Sending such a small current is also how you would braid the major honours to operate your quantum computer. See figure 25.

The major honours are not physically exchanged but the effect of these currents is the same. Now the next step is indeed creating and measuring major owners in our hash-tags networks. This will turn them into major cubed candidates for real quantum computers.

WHAT ITS SIGNIFICANCE?

By making use of these Majorana's it will be possible to use larger number of Qubits on a chip and that's exactly what we need. I suppose that it is comparable to the time we still had the Univac Computers. Big as houses hard to use.
The problem that you need such low temperatures to cool the chips, needs to be solved before we could make use of it in our private homes or carry them around. Maybe another twenty years?


DELPHI THIRD PARTY ALLIANCE
VENDOR ASSOCIATION

FASTREPORT 6.0

THE NEW FASTREPORT 6.0 W
BE RELEASD, SO WHAT'SNEW

What's new FastReport VCL 6.

In the new version FastReport reworked the report objects architecture which allows us to add new objects with new features!

REPORT ENGINE:

- **The new architecture allows to build complicated interactive reports and complicated object-editors which can be used in both the report designer and he preview.**
  With the new object editors users can edit some objects of a prepared report with help of the report designer.

- **Post processing of expressions in «Text» objects.**
  New post processing gives you the ability to calculate expressions inside text objects (and others) by some event (group ending, for example) with delay.
- This mechanism allows to show aggregate functions like Sum at the report, beginning before the total value will be calculated without any script code and just for one data pass.

- **New duplicates processing.**
  With new duplicate-processing-system it is easy to combine duplicate «Text» objects.
  It's possible to clear duplicated text like before, but also to hide objects with the same text and even merge several «Text» objects into one.

- **Transport input-output filters.**
  A New intermediate layer between Save and Load file operations gives the user the ability to easily save and load report templates or prepared reports.
  Also it allows to save export-files to different file storage or send it by e-mail. With the new version it is easy to save report templates, prepared reports or exported results to different places like cloud services or send it by e-mail.
  Delphi's component model allows to easily include filters in the application. With the component-model it's easy to add and control transports in your application.
  FastReport VCL supports the following storage: E-mail, FTP, DropBox, OneDrive, Box.com, GoogleDrive.

NEW OBJECTS:

- The Table object allows you to build a tabular report with variable number of rows and/or columns. With this object it's possible to build complicated tabular reports which do not have frame overlapping. But, first of all Table is designed to make report creation fast and easy. It has rich functionality to manage the table appearance like adding of new Row and Columns, change Row/Column places, join cells, set table dimensions easily, change Row/Column sizes and link cells with data. Just like the "Text" object Table can grow and split.

CHANGE DIMENSION.
MOVE ROWS.
LINK WITH DATA.

- New «Map» Object. You can add geographical maps to your report. The Map Objects supports different maps formats like OSM and ESRI. It has rich abilities like color ranges, highlights, GPX, interactivity and more.
- Gauge object. Add more visual representability and interactivity to the report with new different types of Gauges (interval, linear, radiant and more).

- New barcode types for barcode object Aztec code, MaxiCode and USPS can be used inside the report.

EXPORTS:

- New export filters abilities for PDF, SVG, HTML5 allow to process complicated objects like RichText, Chart, Maps and export them directly as vector/text format approaching WYSIWYG in these formats.

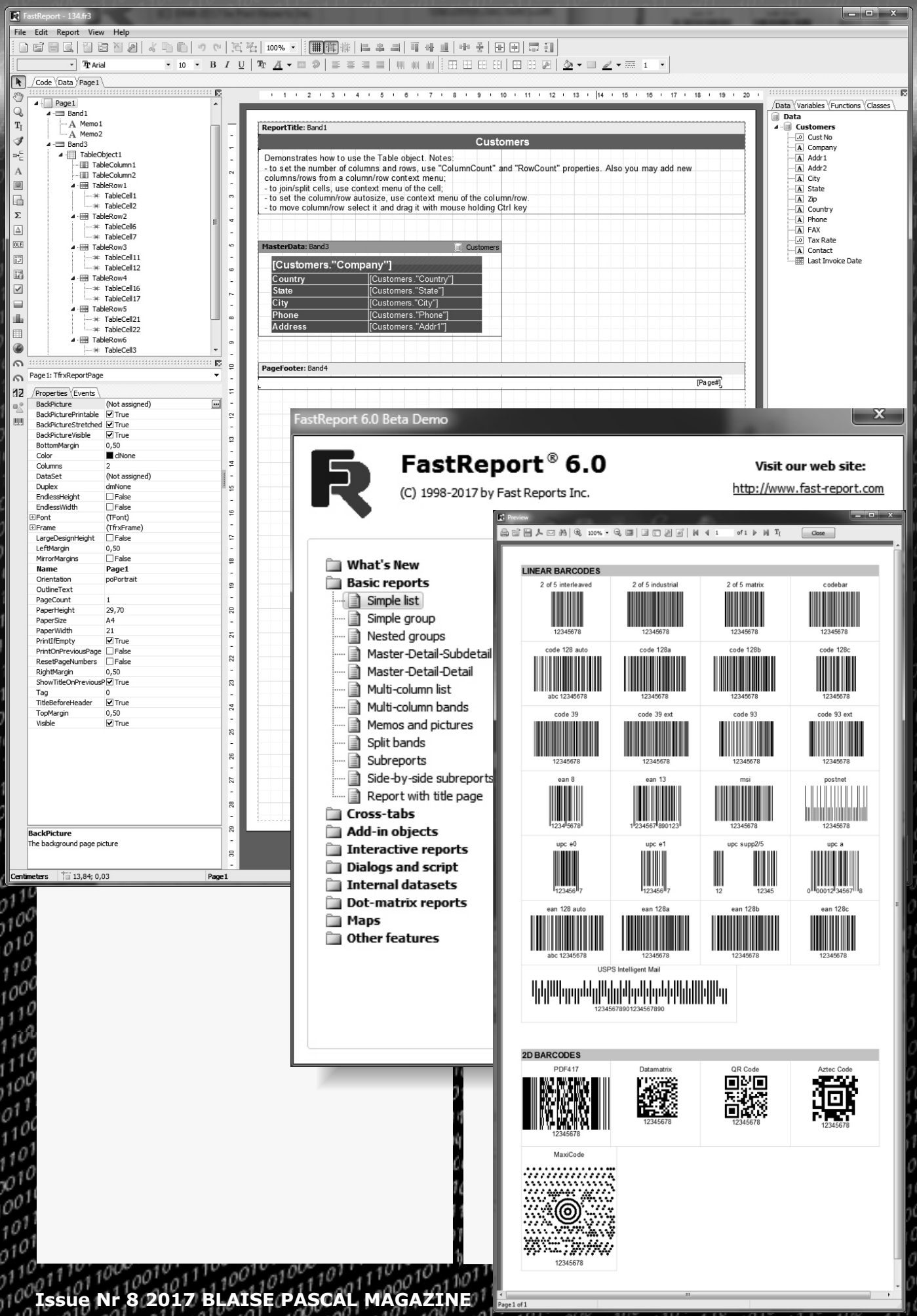
REPORT DESIGNER:

- Improved Guide lines allow to move and resize docked objects. It makes editing of the report easier. Users can set up Guide lines functionality in report designer options.

- Extended script debugger. Improved break points with ability to temporarily disable it and set condition for triggering. New window with "Local" variables list.

Introduction

We want to show in this special article how the most important features of this beautiful program are created and the FR-Demo version gives a very good overview of all the details you are capable of using in conjunction with your FastReport 6.0. The Demo is a very good tool for learning what you can do with fast Report including design examples. That makes it really easy to create reports in your own way. This is a gorgeous tool!!!

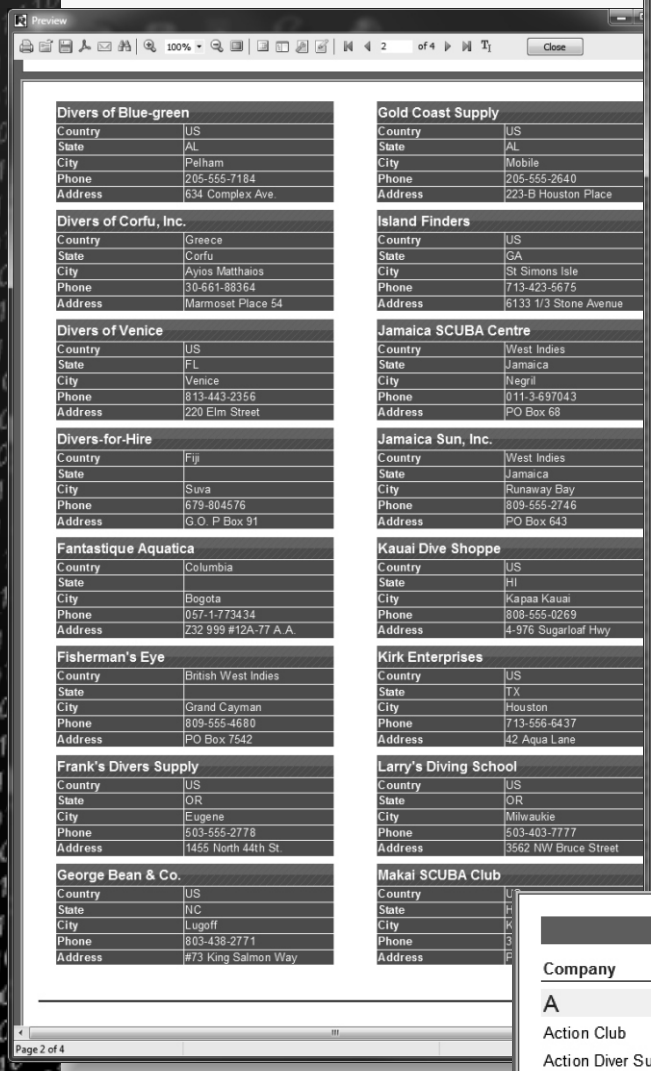

FastReport - 134.fr3
File Edit Report View Help
Code Data Page1
Page1
Band1
Memo1
Memo2
Band3
TableObject1
TableColumn1
TableColumn2
TableRow1
TableCell1
TableCell2
TableRow2
TableCell6
TableCell7
TableRow3
TableCell11
TableCell12
TableRow4
TableCell16
TableCell17
TableRow5
TableCell21
TableCell22
TableRow6
TableCell3
Page1: TfrxReportPage
Properties Events
BackPicture (Not assigned)
BackPicturePrintable True
BackPictureStretched True
BackPictureVisible True
BottomMargin 0,50
Color clNone
Columns 2
DataSet (Not assigned)
Duplex dmNone
EndlessHeight False
EndlessWidth False
Font (TFont)
Frame (TfrxFrame)
LargeDesignHeight False
LeftMargin 0,50
MirrorMargins False
Name Page1
Orientation poPortrait
OutlineText
PageCount 1
PaperHeight 29,70
PaperSize A4
PaperWidth 21
PrintIfEmpty True
PrintOnPreviousPage False
ResetPageNumbers False
RightMargin 0,50
ShowTitleOnPreviousP True
Tag 0
TitleBeforeHeader True
TopMargin 0,50
Visible True
BackPicture
The background page picture
Centimeters 13,84; 0,03 Page1
ReportTitle: Band1
Customers
Demonstrates how to use the Table object. Notes:
- to set the number of columns and rows, use "ColumnCount" and "RowCount" properties. Also you may add new columns/rows from a column/row context menu;
- to join/split cells, use context menu of the cell;
- to set the column/row autosize, use context menu of the column/row.
- to move column/row select it and drag it with mouse holding Ctrl key
MasterData: Band3 Customers
[Customers."Company"]
Country [Customers."Country"]
State [Customers."State"]
City [Customers."City"]
Phone [Customers."Phone"]
Address [Customers."Addr1"]
PageFooter: Band4
[Page#]
Data Variables Functions Classes
Data
Customers
Cust No
Company
Addr1
Addr2
City
State
Zip
Country
Phone
FAX
Tax Rate
Contact
Last Invoice Date
FastReport 6.0 Beta Demo
FastReport® 6.0
(C) 1998-2017 by Fast Reports Inc.
Visit our web site:
http://www.fast-report.com
What's New
Basic reports
Simple list
Simple group
Nested groups
Master-Detail-Subdetail
Master-Detail-Detail
Multi-column list
Multi-column bands
Memos and pictures
Split bands
Subreports
Side-by-side subreports
Report with title page
Cross-tabs
Add-in objects
Interactive reports
Dialogs and script
Internal datasets
Dot-matrix reports
Maps
Other features
Preview
Close
LINEAR BARCODES
2 of 5 interleaved
2 of 5 industrial
2 of 5 matrix
codebar
code 128 auto
code 128a
code 128b
code 128c
code 39
code 39 ext
code 93
code 93 ext
ean 8
ean 13
msi
postnet
upc e0
upc e1
upc supp2/5
upc a
ean 128 auto
ean 128a
ean 128b
ean 128c
USPS Intelligent Mail
2D BARCODES
PDF417
Datamatrix
QR Code
Aztec Code
MaxiCode
Page 1 of 1

Demonstrates how to use the Table object. Notes:

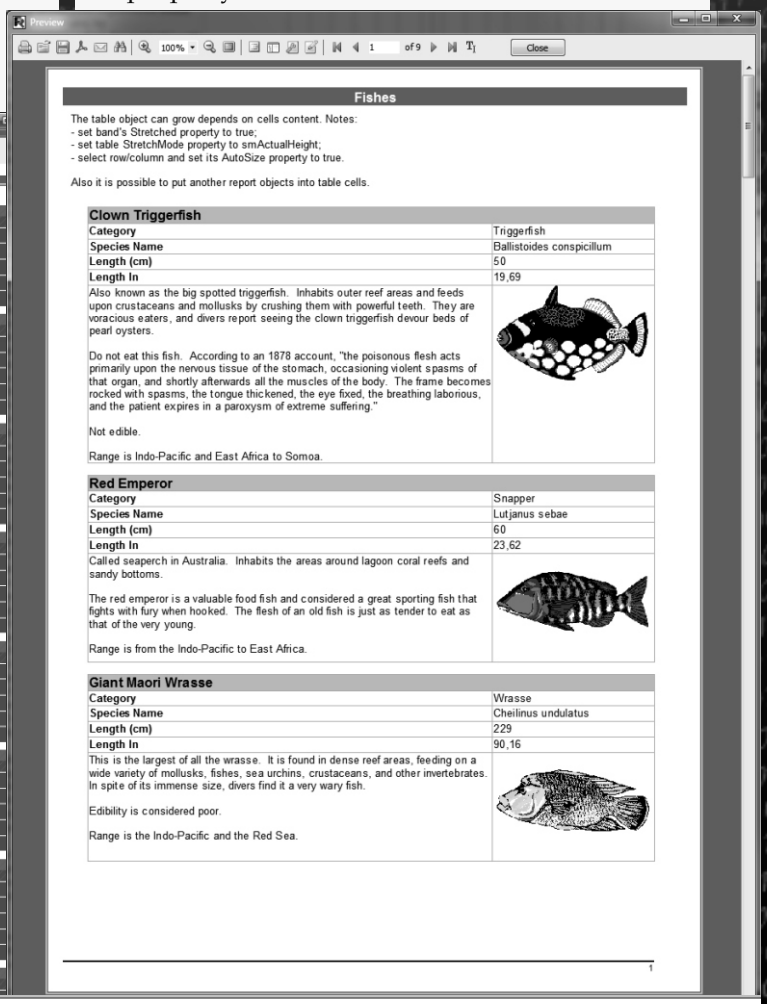
- to set the number of columns and rows, use "ColumnCount" and "RowCount". properties. Also you may add new columns/rows from a column/ row context menu;
- to join/split cells, use context menu of the cell;
- to set the column/row autosize, use context menu of the column/row.

The table object can grow depends on cells content. Notes:

- set band's Stretched property to true;
- set table StretchMode property to `smActualHeight;`
- select row/column and set its AutoSize property to true.



Divers of Blue-green	
Country	US
State	AL
City	Pelham
Phone	205-555-7184
Address	634 Complex Ave.
Gold Coast Supply	
Country	US
State	AL
City	Mobile
Phone	205-555-2640
Address	223-B Houston Place
Divers of Corfu, Inc.	
Country	Greece
State	Corfu
City	Ayios Matthaios
Phone	30-661-88364
Address	Marmoset Place 54
Island Finders	
Country	US
State	GA
City	St Simons Isle
Phone	713-423-5675
Address	6133 1/3 Stone Avenue
Divers of Venice	
Country	US
State	FL
City	Venice
Phone	813-443-2356
Address	220 Elm Street
Jamaica SCUBA Centre	
Country	West Indies
State	Jamaica
City	Negril
Phone	011-3-697043
Address	PO Box 68
Divers-for-Hire	
Country	Fiji
State	
City	Suva
Phone	679-804576
Address	G.O. P Box 91
Jamaica Sun, Inc.	
Country	West Indies
State	Jamaica
City	Runaway Bay
Phone	809-555-2746
Address	PO Box 643
Fantastique Aquatica	
Country	Columbia
State	
City	Bogota
Phone	057-1-773434
Address	Z32 999 #12A-77 A.A.
Kauai Dive Shoppe	
Country	US
State	HI
City	Kapaa Kauai
Phone	808-555-0269
Address	4-976 Sugarloaf Hwy
Fisherman's Eye	
Country	British West Indies
State	
City	Grand Cayman
Phone	809-555-4680
Address	PO Box 7542
Kirk Enterprises	
Country	US
State	TX
City	Houston
Phone	713-556-6437
Address	42 Aqua Lane
Frank's Divers Supply	
Country	US
State	OR
City	Eugene
Phone	503-555-2778
Address	1455 North 44th St.
Larry's Diving School	
Country	US
State	OR
City	Milwaukie
Phone	503-403-7777
Address	3562 NW Bruce Street
George Bean & Co.	
Country	US
State	NC
City	Lugoff
Phone	803-438-2771
Address	#73 King Salmon Way
Makai SCUBA Club	
Country	U
State	H
City	K
Phone	3
Address	P



Fishes

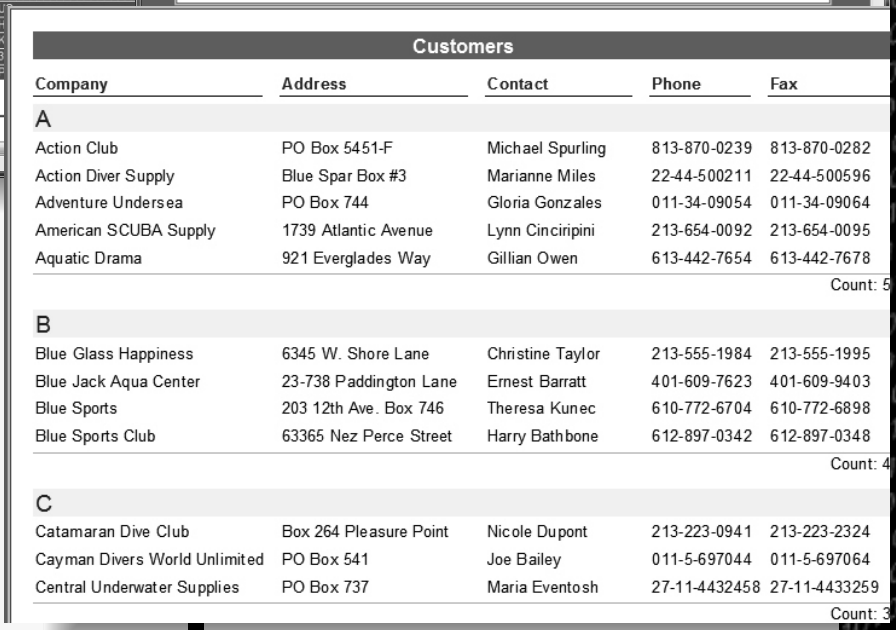
The table object can grow depends on cells content. Notes:
- set band's Stretched property to true;
- set table StretchMode property to smActualHeight;
- select row/column and set its AutoSize property to true.

Also it is possible to put another report objects into table cells.

Clown Triggerfish	
Category	Triggerfish
Species Name	Ballistoides conspicillum
Length (cm)	50
Length In	19,69
Also known as the big spotted triggerfish. Inhabits outer reef areas and feeds upon crustaceans and mollusks by crushing them with powerful teeth. They are voracious eaters, and divers report seeing the clown triggerfish devour beds of pearl oysters. Do not eat this fish. According to an 1878 account, "the poisonous flesh acts primarily upon the nervous tissue of the stomach, occasioning violent spasms of that organ, and shortly afterwards all the muscles of the body. The frame becomes rocked with spasms, the tongue thickened, the eye fixed, the breathing labonous, and the patient expires in a paroxysm of extreme suffering." Not edible. Range is Indo-Pacific and East Africa to Somoa.	
Red Emperor	
Category	Snapper
Species Name	Lutjanus sebae
Length (cm)	60
Length In	23,62
Called seaperch in Australia. Inhabits the areas around lagoon coral reefs and sandy bottoms. The red emperor is a valuable food fish and considered a great sporting fish that fights with fury when hooked. The flesh of an old fish is just as tender to eat as that of the very young. Range is from the Indo-Pacific to East Africa.	
Giant Maori Wrasse	
Category	Wrasse
Species Name	Cheilinus undulatus
Length (cm)	229
Length In	90,16
This is the largest of all the wrasse. It is found in dense reef areas, feeding on a wide variety of mollusks, fishes, sea urchins, crustaceans, and other invertebrates. In spite of its immense size, divers find it a very wary fish. Edibility is considered poor. Range is the Indo-Pacific and the Red Sea.	

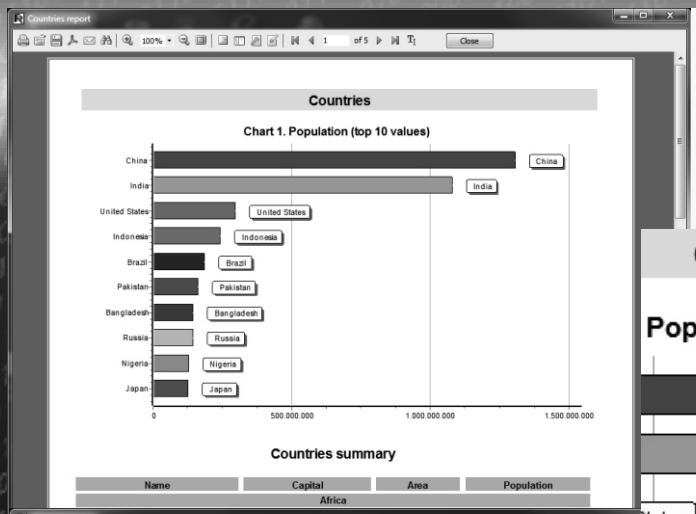
Demonstrates the "Duplicates" feature. To use it:

- select a Text object that prints duplicate values;
- set its "Duplicates" property to the desired value:
- "Show" will show duplicates;
- "Hide" will hide the object;
- "Clear" will clear the object's text but show the object;
- "Merge" will merge objects with the same values.

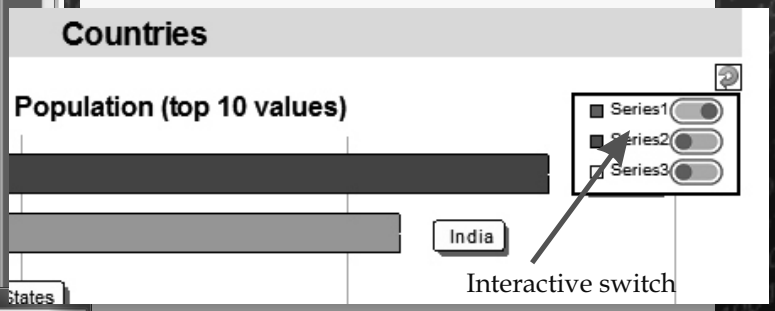


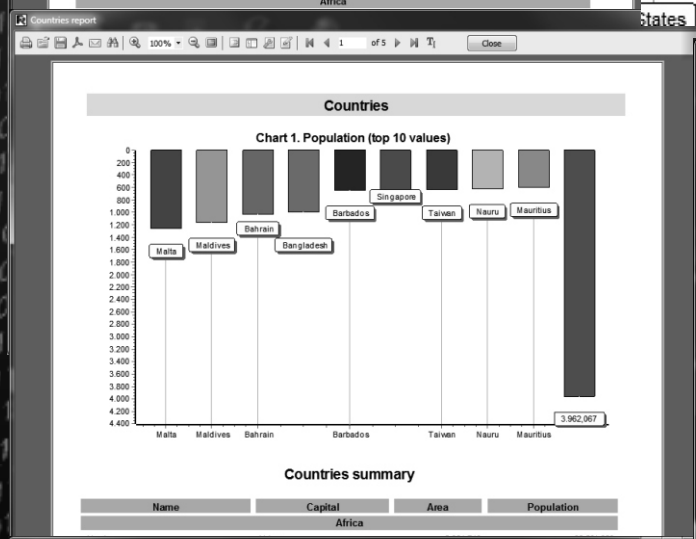
Customers

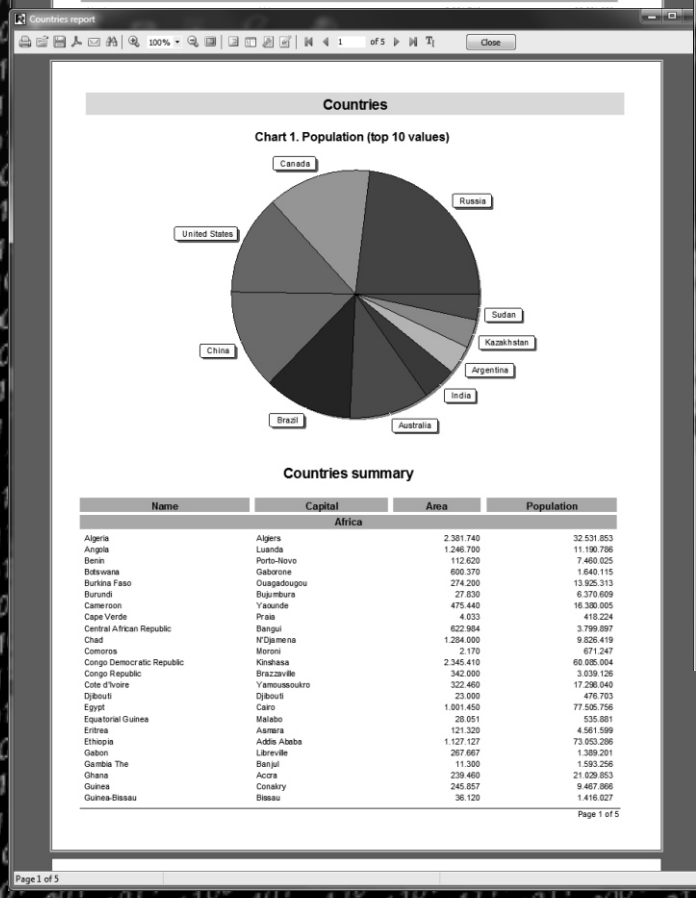
Company	Address	Contact	Phone	Fax
A				
Action Club	PO Box 5451-F	Michael Spurling	813-870-0239	813-870-0282
Action Diver Supply	Blue Spar Box #3	Marianne Miles	22-44-500211	22-44-500596
Adventure Undersea	PO Box 744	Gloria Gonzales	011-34-09054	011-34-09064
American SCUBA Supply	1739 Atlantic Avenue	Lynn Cinciripini	213-654-0092	213-654-0095
Aquatic Drama	921 Everglades Way	Gillian Owen	613-442-7654	613-442-7678
				Count: 5
B				
Blue Glass Happiness	6345 W. Shore Lane	Christine Taylor	213-555-1984	213-555-1995
Blue Jack Aqua Center	23-738 Paddington Lane	Ernest Barratt	401-609-7623	401-609-9403
Blue Sports	203 12th Ave. Box 746	Theresa Kunec	610-772-6704	610-772-6898
Blue Sports Club	63365 Nez Perce Street	Harry Bathbone	612-897-0342	612-897-0348
				Count: 4
C				
Catamaran Dive Club	Box 264 Pleasure Point	Nicole Dupont	213-223-0941	213-223-2324
Cayman Divers World Unlimited	PO Box 541	Joe Bailey	011-5-697044	011-5-697064
Central Underwater Supplies	PO Box 737	Maria Eventosh	27-11-4432458	27-11-4433259
				Count: 3

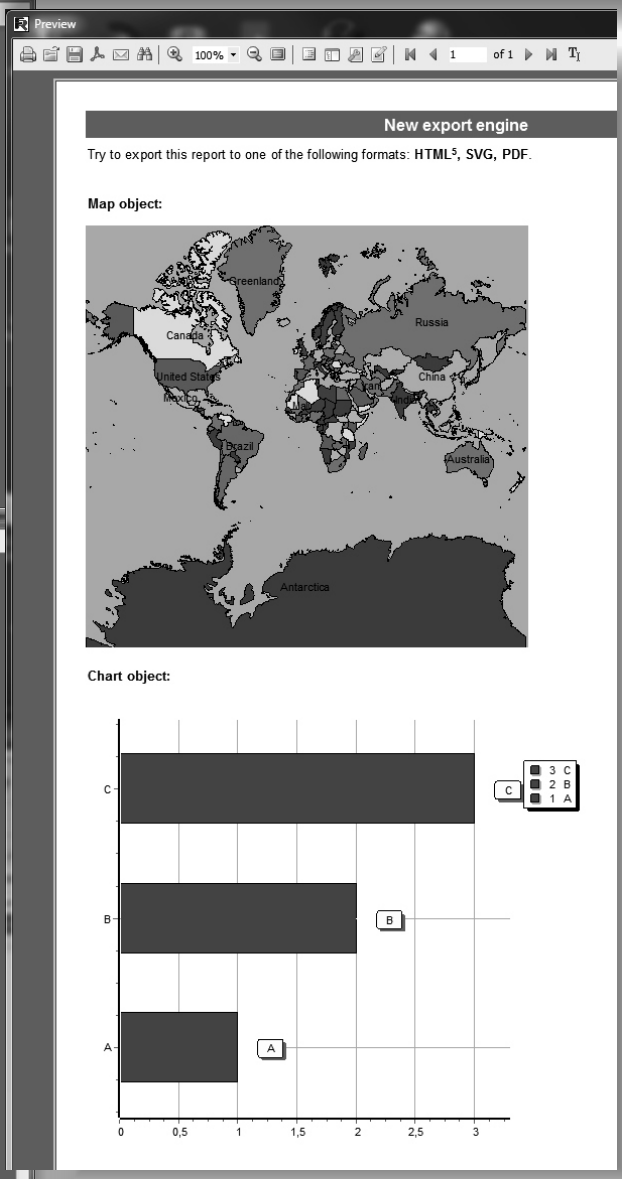


New architecture allows to build complicated interactive reports and complicated objects editors which can be used in both report designer and preview. With the new object editors users can edit some objects of prepared report with the report designer.







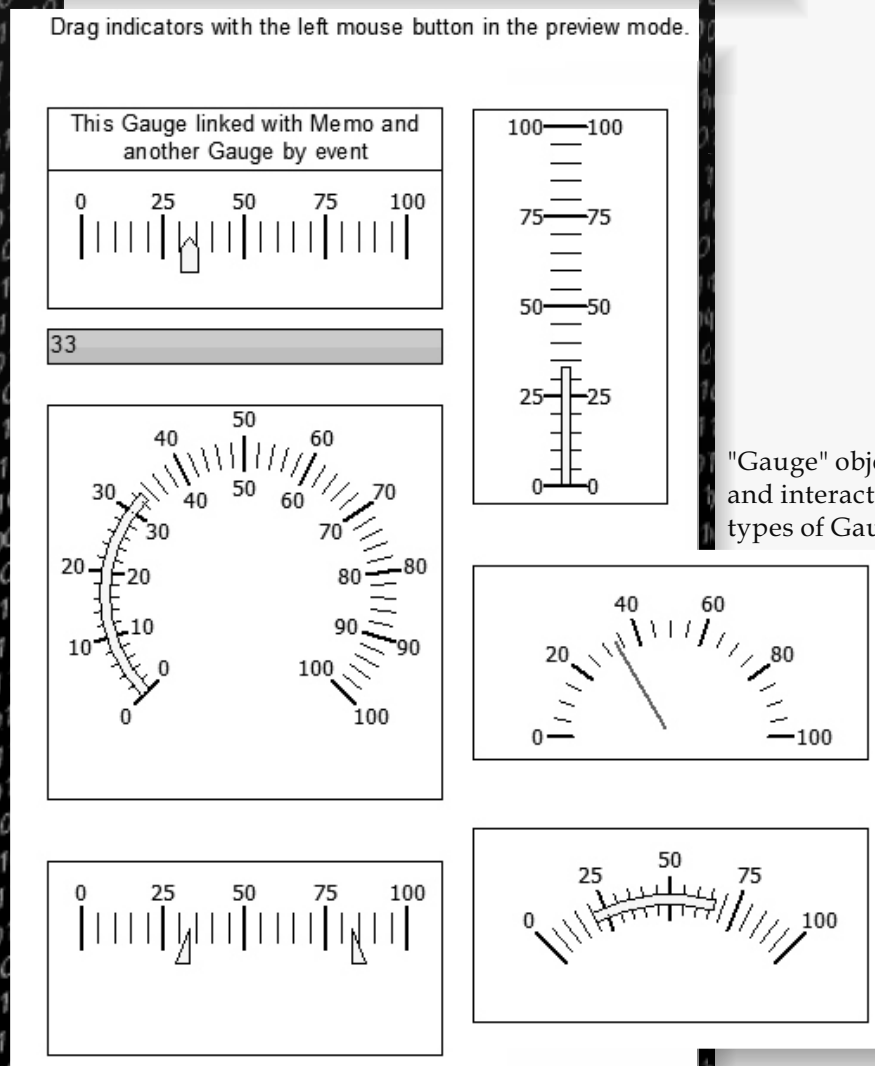


Demonstrates export of complicated objects to vector format. Try to export this report to one of the following formats: HTML 5, SVG, PDF.

Demonstrates the Map object with two layers. The first layer displays a static world map and is not bound to a data. The second layer displays application provided spatial data in format of latitude/longitude/name/value.

To create a map:

- put the Map object on the report page;
- right-click the object, to pop up its menu;
- select "Add Layer" to add a map layer;
- select ESRI MapFile (.shp/.dbf files);
- right-click the object again, select "Add Layer", and select empty application layer;
- set the dataset for the map second layer;
- set the spatial data in format of latitude/longitude/name (*in this example the name of a city*) and the analytical data (*in this example an expression that returns the sales in this city*);



"Gauge" object. Add more visual representability and interactivity to the report with new different types of Gauges (*interval, linear, radiant and more*).

Preview

Customers

Invoice total for all customers: € 3.093.011,75

Company	Phone	Fax	
Action Club	813-870-0239	813-870-0282	€ 31.399,65

Order No 1014 Date 25-5-1988 Total this order: € 134,85 from € 31.399,65

Part	Description	Price	Qty
7612	Krypton Flashlight	44,95	3

Total this order

Order No 1029 Date 18-7-1988 Total this order: € 2

Part	Description	Price	Qty
1328	Regulator System	430	46
5313	Safety Knife	41	8

Total this order:

Order No 1038 Date 26-8-1988 Total this order: € 1

Part	Description	Price	Qty
1986	Depth/Pressure Gauge Console	188	54

Total this order:

Order No 1129 Date 19-10-1993 Total this order: €

Part	Description	Price	Qty
7619	Flashlight (Rechargeable)	169,95	4
9316	95.1 cu ft Tank	325	1

Total this orde

Total sales this customer: € 31.399,65

Company	Phone	Fax
Action Diver Supply	22-44-500211	22-44-500596

Order No 1039 Date 29-8-1988 Total this order:

Part	Description	Price	Qty
5318	Medium Titanium Knife	56,95	4
1946	Second Stage Regulator	309	1

Total this or

Total sales this customer: € 536,80

Company	Phone	Fax
Adventure Undersea	011-34-09054	011-34-09064

Order No 1017 Date 12-6-1988 Total this order: € 1

Part	Description	Price	Qty
2350	Compass Console Mount	29	5
1986	Depth/Pressure Gauge Console	188	45
2341	Depth/Pressure Gauge	105	4
2314	Electronic Console	390	3

Total this order:

Order No 1037 Date 26-8-1988 Total this order: €

Part	Description	Price	Qty
3316	Stabilizing Vest	430	6
9312	60.6 cu ft Tank	179	3

Total this orde

Order No 1074 Date 19-4-1989 Total this order: €

Part	Description	Price	Qty
12310	Sonar System	439	5

Total this orde

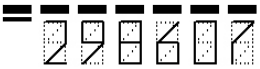
Order No 1099 Date 16-6-1989 Total this order:

Part	Description	Price	Qty

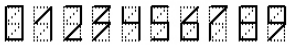
This report shows how to use late object processing and print SUM in the group header bands with different groups levels.

Three-digit code stamp for international shipments

Six-digit code stamp for local shipments



Sample of writing index digits



Custom code stamps

2619	Navigation Compass	19,95	1
3386	Welded Seam Stabilizing Vest	280	3

Total this order: € 859,95

Order No 1117 Date 13-4-1993 Total this order: € 6.734,85 from € 124.983,95

Part	Description	Price	Qty
5349	Flashlight	65	15
2341	Depth/Pressure Gauge	105	2
7619	Flashlight (Rechargeable)	169,95	3
912	Underwater Diver Vehicle	1680	3

Total this order: € 6.734,85

Order No 1137 Date 27-11-1993 Total this order: € 6.785,40 from € 124.983,95

Part	Description	Price	Qty
1330	Alternate Inflation Regulator	260	17
2954	Dive Computer	650	1
3386	Welded Seam Stabilizing Vest	280	1
7654	Halogen Flashlight	59,95	12
9312	60.6 cu ft Tank	179	4

Total this order: € 6.785,40

Order No 1217 Date 22-11-1994 Total this order: € 84.219,90 from € 124.983,95

Part	Description	Price	Qty
900	Dive kayak	3999,95	18
1390	First Stage Regulator	170	3
2390	Electronic Console w/ options	420	3
2648	Depth/Pressure Gauge (Analog)	119	16
11652	Video Light	359,95	4
11221	Remotely Operated Video System	2369	3

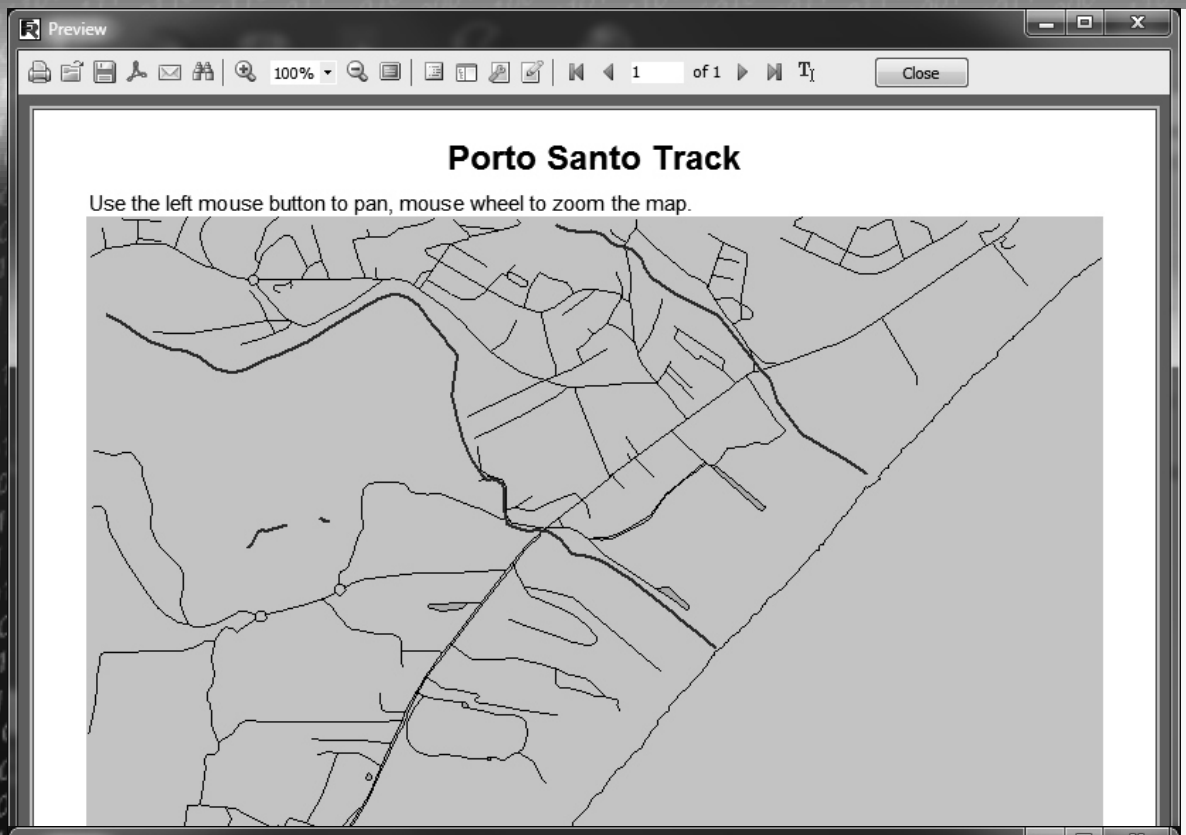
Total this order: € 84.219,90

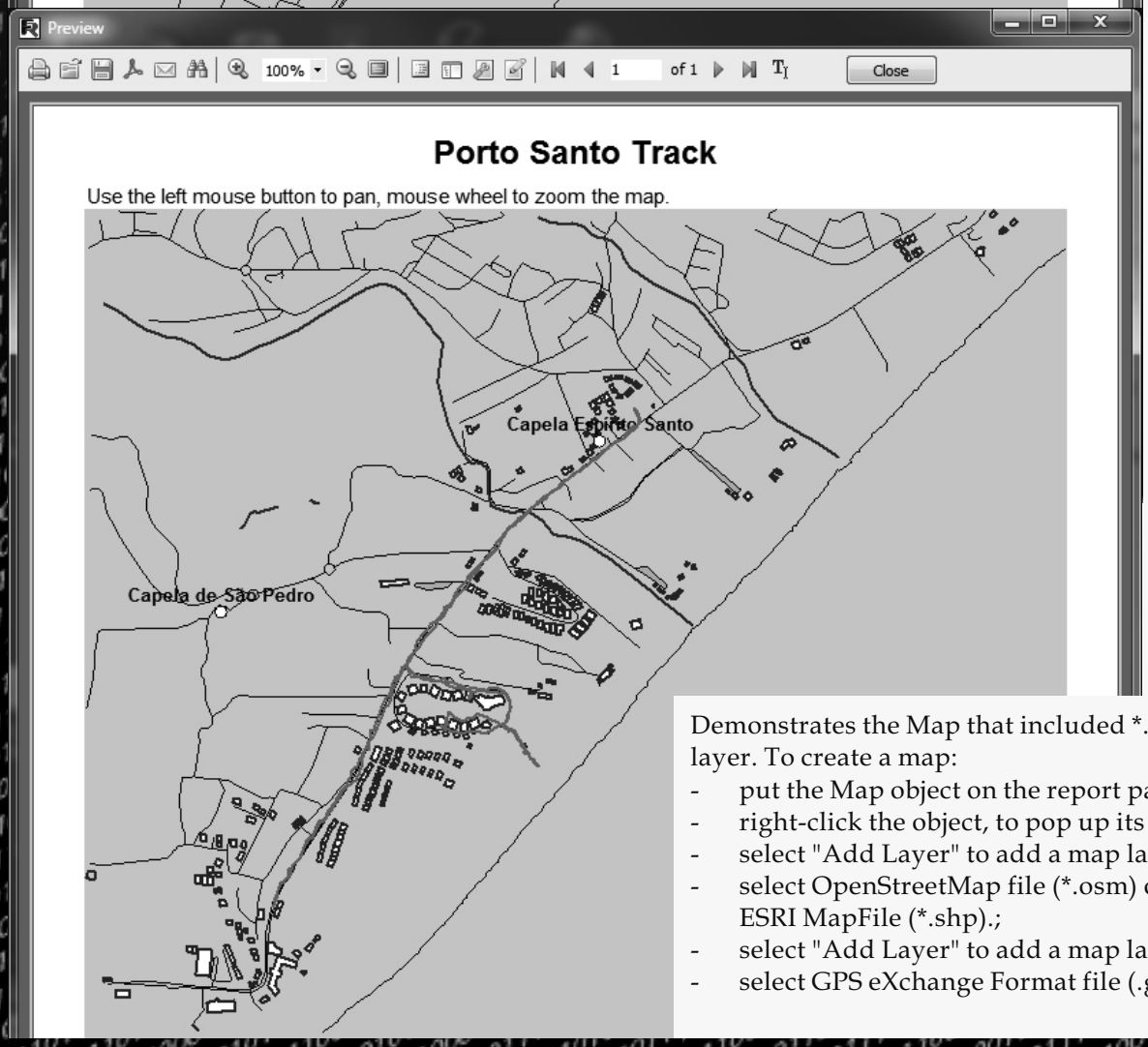
Order No 1294 Date 4-1-1995 Total this order: € 3.304,85 from € 124.983,95

Part	Description	Price	Qty

Page 1 of 33

New barcode types for barcode object Aztec code, MaxiCode and USPS can be used inside the report.



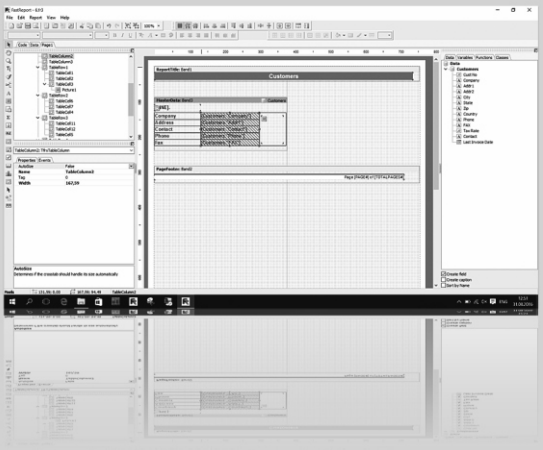


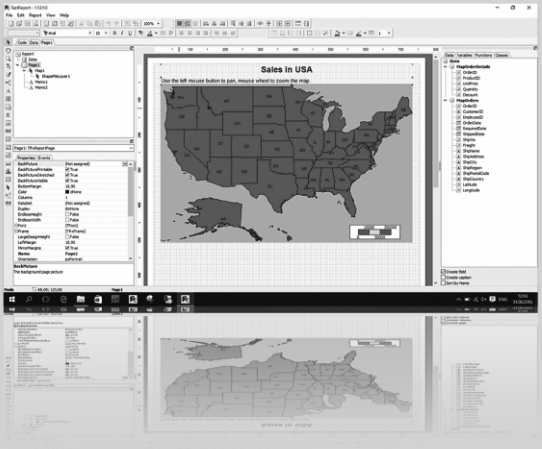
Demonstrates the Map that included *.gpx track layer. To create a map:

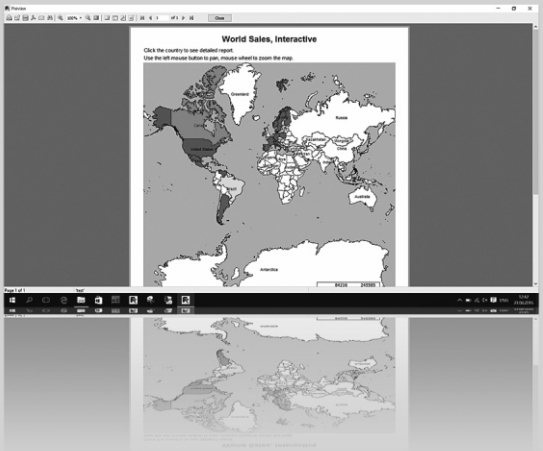
- put the Map object on the report page;
- right-click the object, to pop up its menu;
- select "Add Layer" to add a map layer;
- select OpenStreetMap file (*.osm) or ESRI MapFile (*.shp).;
- select "Add Layer" to add a map layer;
- select GPS eXchange Format file (.gpx).

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

New **FastReport® VCL 6** is coming!







New objects:

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

Report engine:

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

Export engine:

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

Fast Reports
Reporting must be Fast!

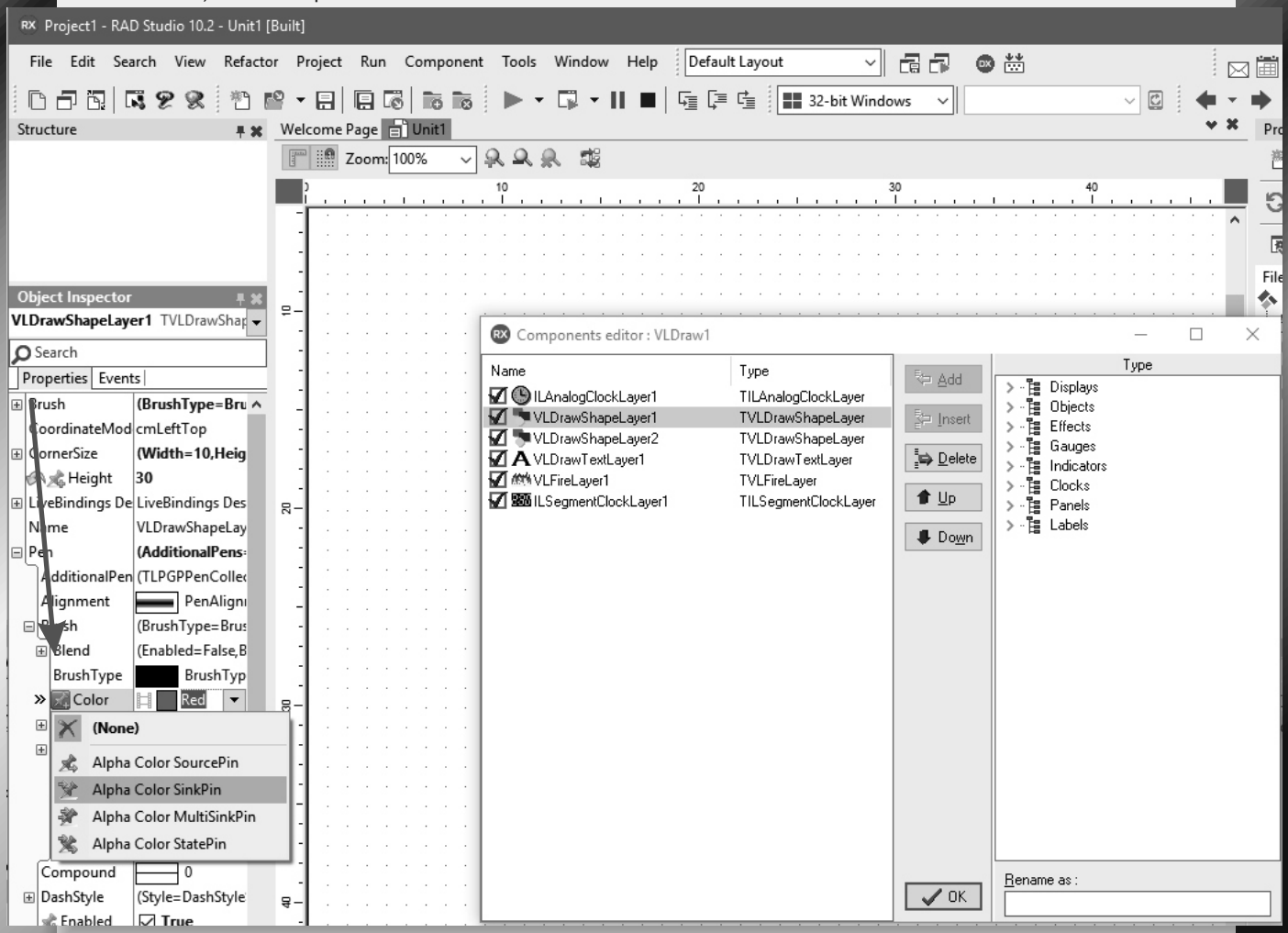
starter expert DX Delphi

INTRODUCTION

In the previous Article, I showed you how to add video layers to the video, and how to apply effects such as fire on them. I demonstrated few different types of layers, some of them simple shapes, others as complex as visual instruments.
Adding video layers and effects is great, but they can be made even more exciting by adding animations.
In this article I will show you how to animate the layers with the TimeLine animation component from AnimationLab.

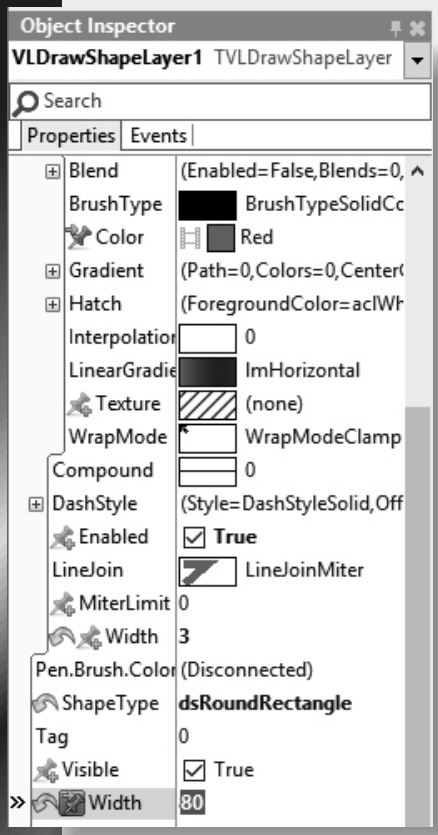
To animate the properties, we will use **Animation TimeLine Component**.
Double click on the VLDraw1 component to open the **Components editor**.
In the left view of the Components editor, select the **VLDrawShapeLayer1** component.
In the Object Inspector expand the "Pen" property and then expand the "Brush" sub property of the "Pen" property.
Select the "Color" sub property of the "Brush" sub property.

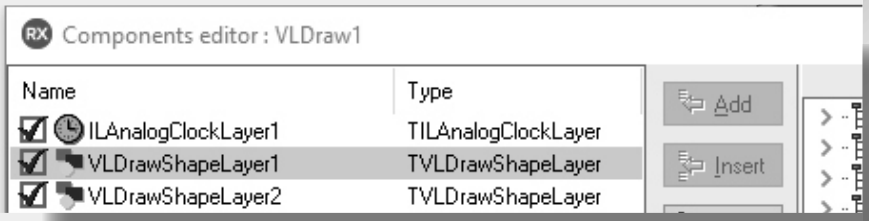
Click on the button at front of the property name to open the pin live binding menu.
In the menu, select "Alpha Color SinkPin":



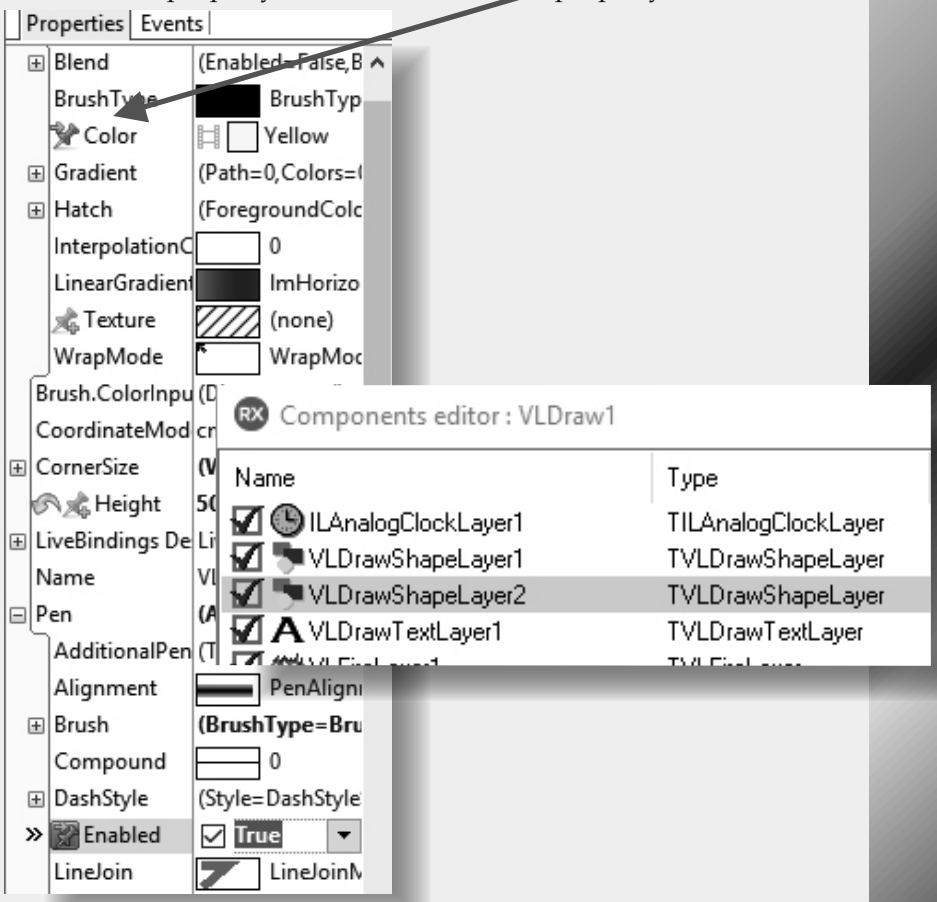
This will add Input Pin for the Color property.

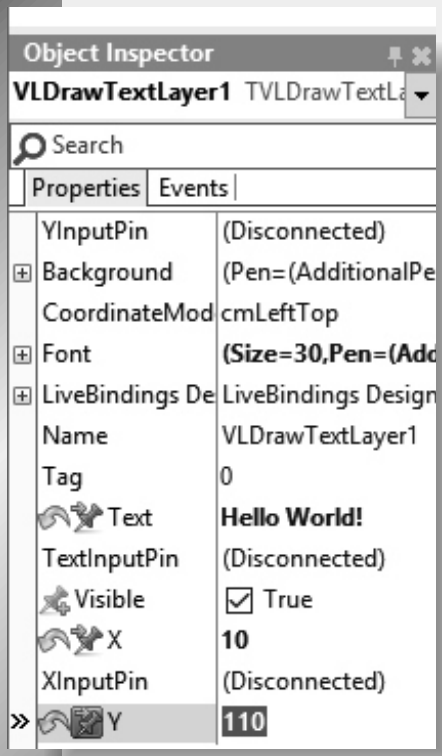
In the Object Inspector select the "Width" property of the **VLDrawShapeLayer1** component.
Click on the button at front of the property name to open the pin live binding menu.
In the menu, select "Integer SinkPin":



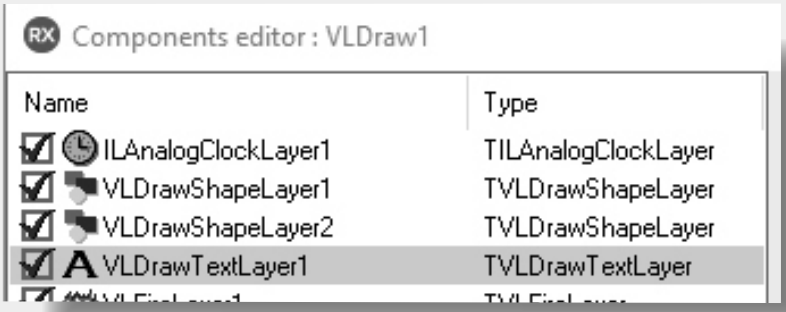


In the left view of the **Components editor**, select the **VLDrawShapeLayer2** component.
In the **Object Inspector**, the same way, add **Sink Pins** for the "Brush.Color" property, and the "Pen.Enabled" property:

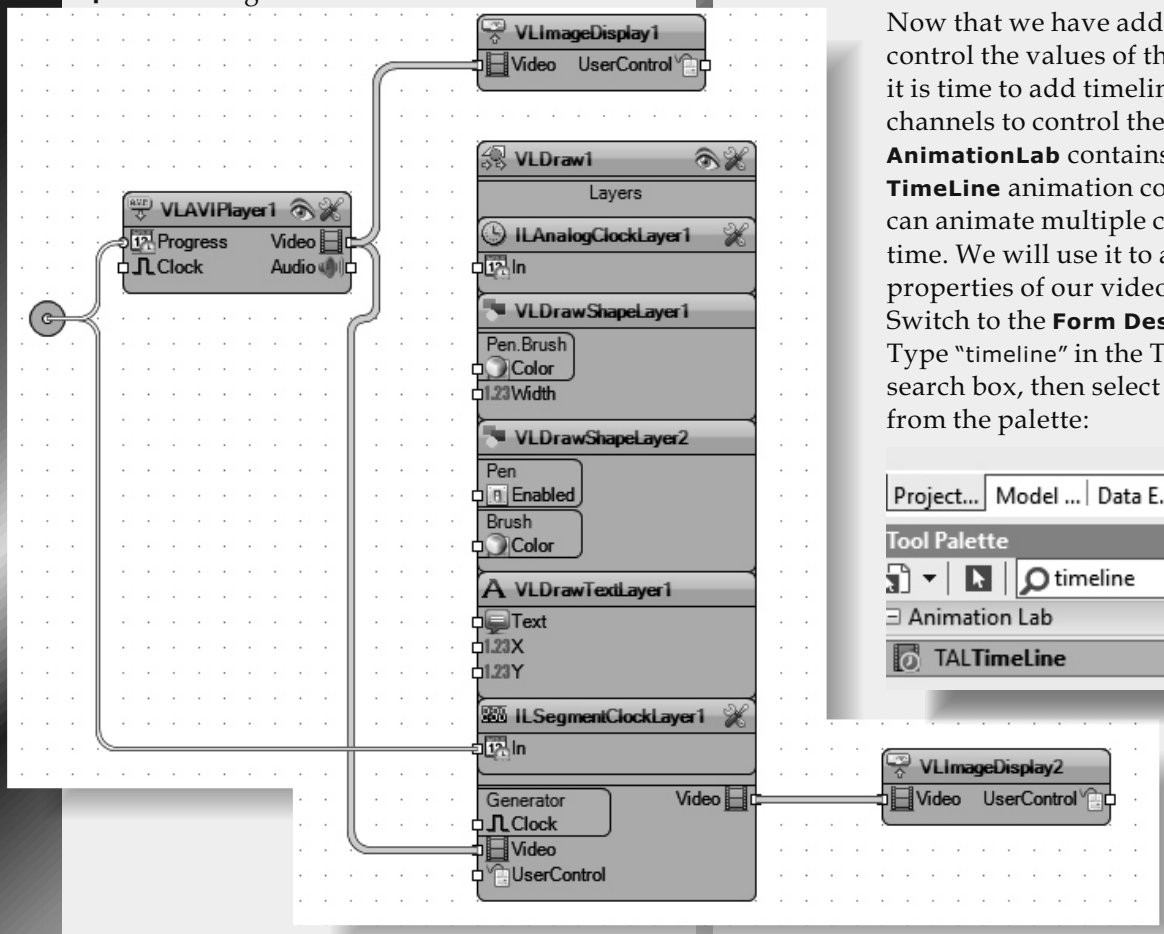




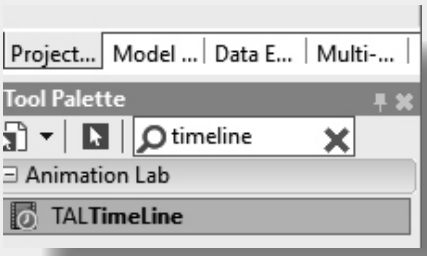
In the left view of the **Components editor,** select the **VLDrawTextLayer1** component.
In the Object Inspector, add Sink Pins for the "Text" property, the "X" property, and the "Y" property:



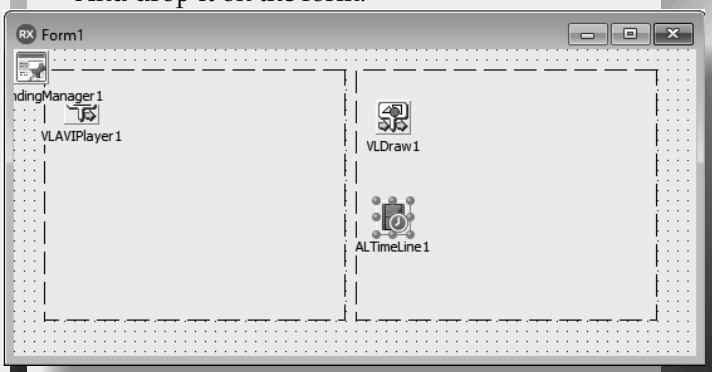
When you add the pins in the **Object Inspector,** they will appear on the components in the **OpenWire** diagram:



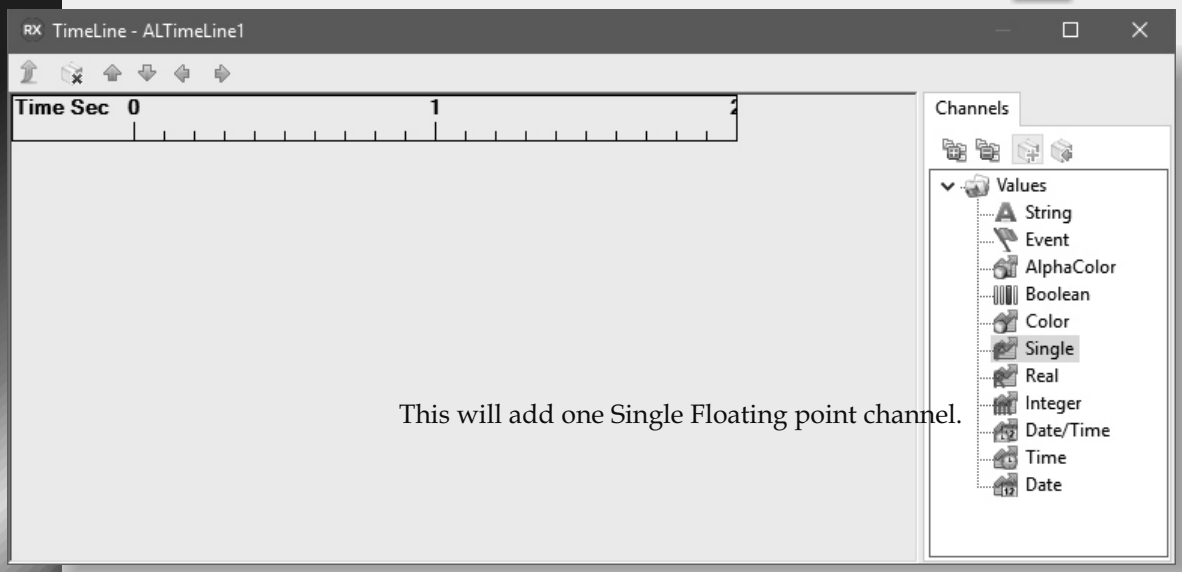
Now that we have added the pins to control the values of the properties, it is time to add timeline with channels to control them over time. **AnimationLab** contains a powerful **TimeLine** animation component that can animate multiple channels over time. We will use it to animate the properties of our video layers. Switch to the **Form Designer.** Type "timeline" in the Tool Palette search box, then select **TALTimeLine** from the palette:



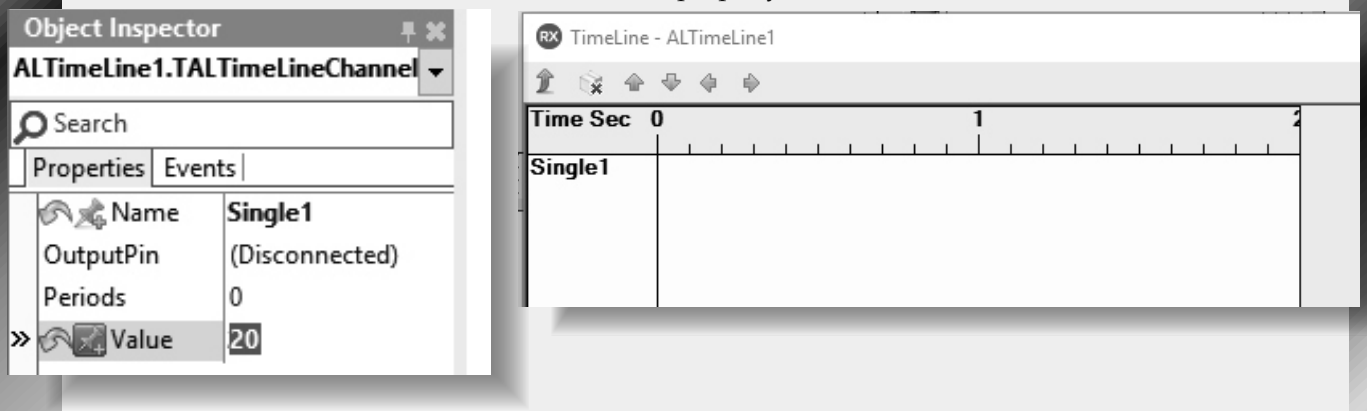
And drop it on the form:



Double click on the component to open the **Animation Timeline Editor**.
The TimeLine can have many channels of different types. First we will add 2 **Floating Point Single** precision channels to animate the X and Y position of the text in the video.
In the right view of the editor select "Single" channel, and then click on the button:

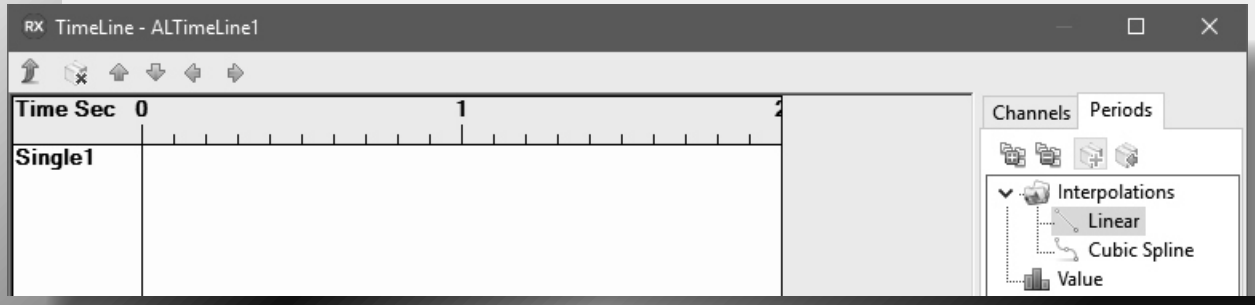


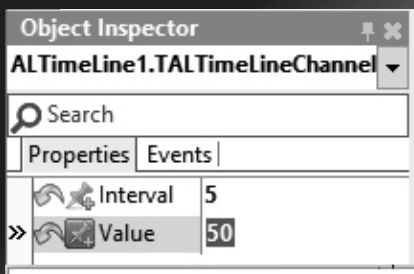
In the **Object Inspector** set the value of the "Value" property to "20":



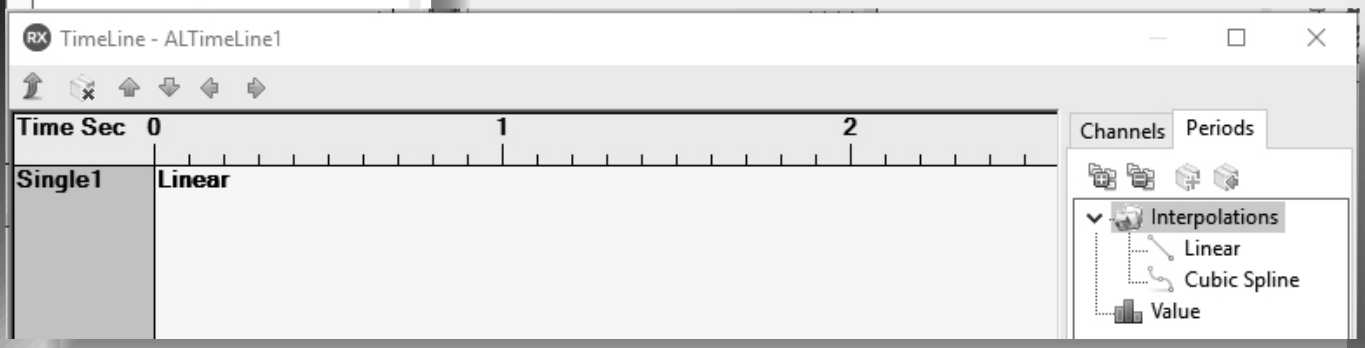
This will be the initial value of the channel.
When a channel is selected in the **TimeLine editor**, a new "Periods" tab will appear on the right.
In this Tab, you can add different types of periods to the timeline. The types will differ for different channels depending on the channel type.
For floating point channels you can add "Linear", "Cubic Spline" or "Value" periods.
The "Linear" period will change the channel value over time linearly.
The **"Cubic Spline"** period will change it using a multiple points Cubic Spline.
And the "Value" period will change it to new value at the end of the period.
We will start by adding a "Linear" period.

In the right view of the editor select "Linear" period, and then click on the button:

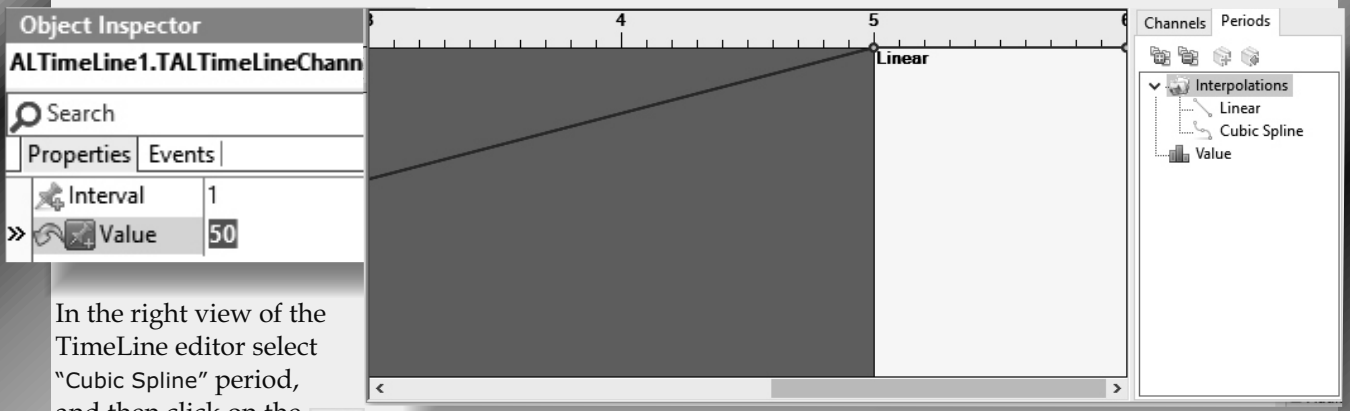




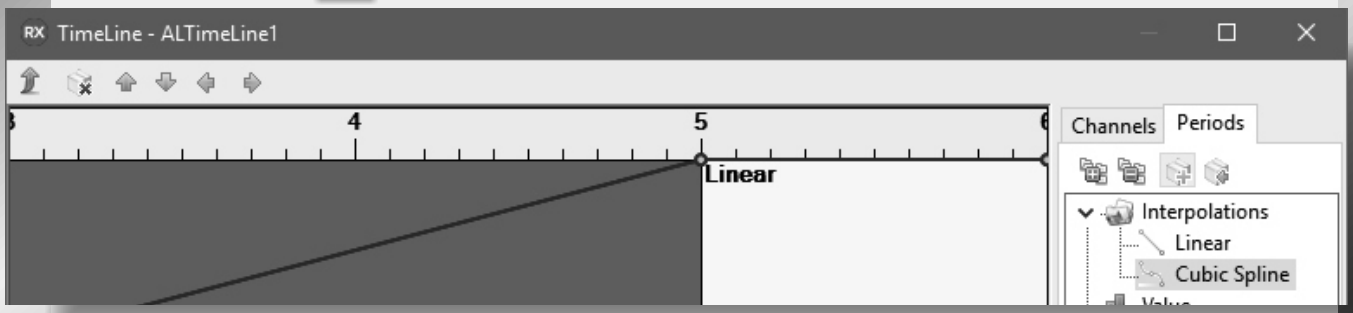
In the **Object Inspector** set the value of the "Interval" property to "5". This will specify that the interval will be 5 seconds long. In the **Object Inspector** set the value of the "Value" property to "50":



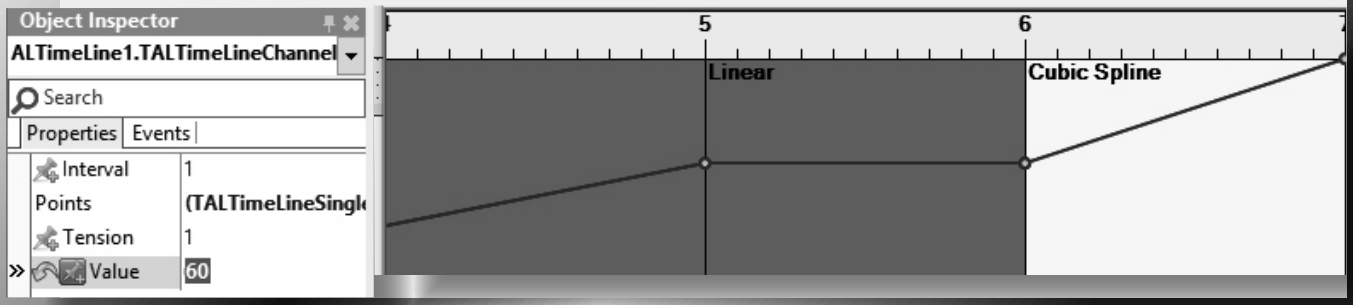
Add a second "Linear" period. In the **Object Inspector** again set the value of the "Value" property to "50":



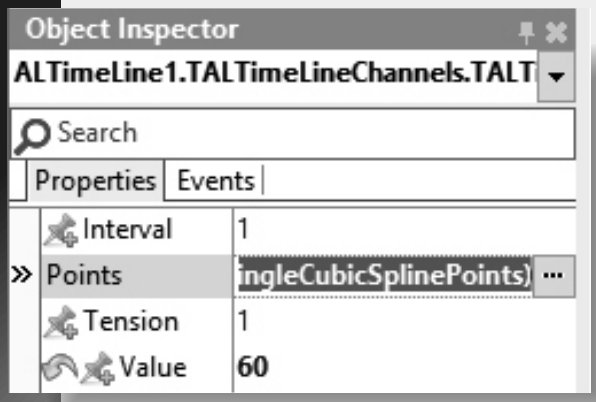
In the right view of the TimeLine editor select "Cubic Spline" period, and then click on the button:



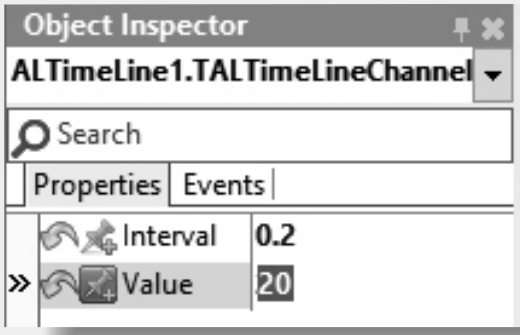
In the **Object Inspector** again set the value of the "Value" property to "60":



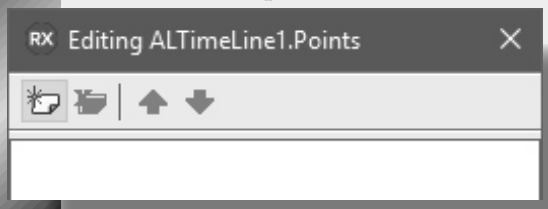
In the **Object Inspector** select the "Points" property, and click on the **"..."** (Ellipsis) button:

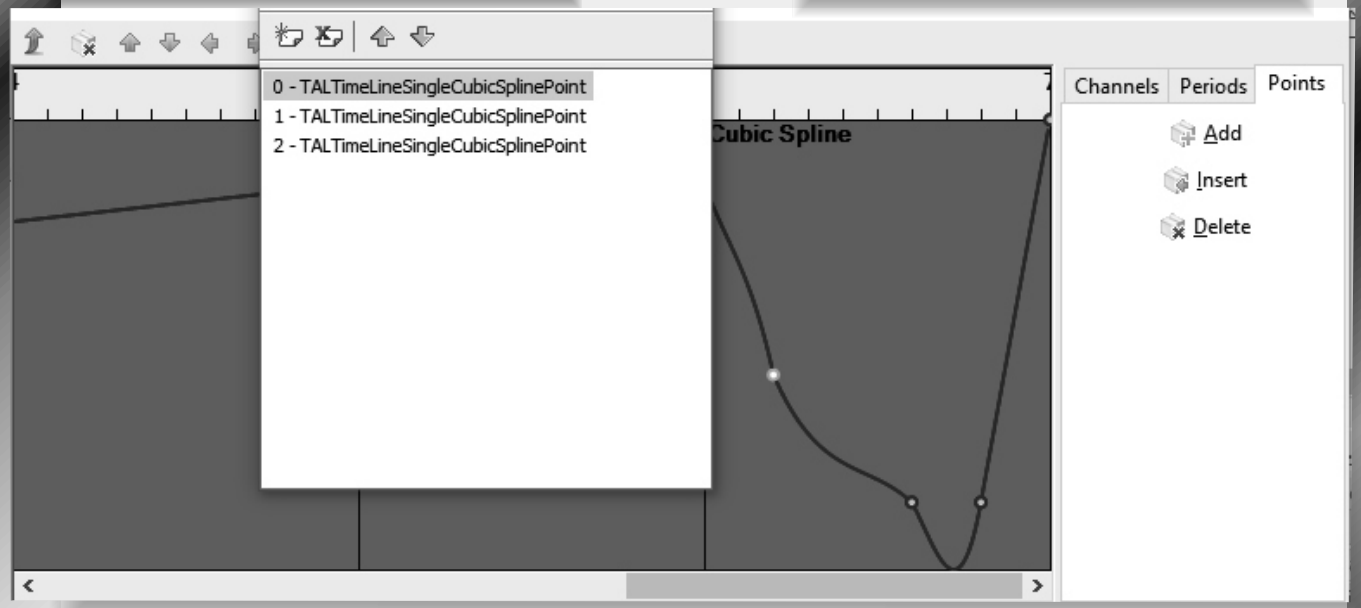


Select the first point.
In the **Object Inspector** set the value of the "Interval" property to "0.2". The Interval of the Cubic Spline points should be between 0 and 1, and is a fraction of the total period interval.
In the Object Inspector set the value of the "Value" property to "20":

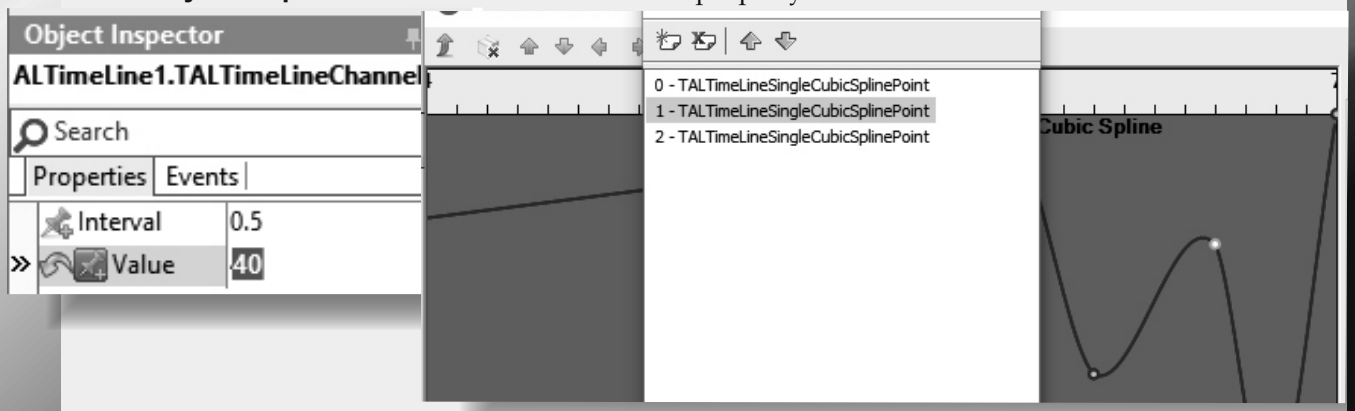


In the collection editor, click 3 times on the button to add 3 points:

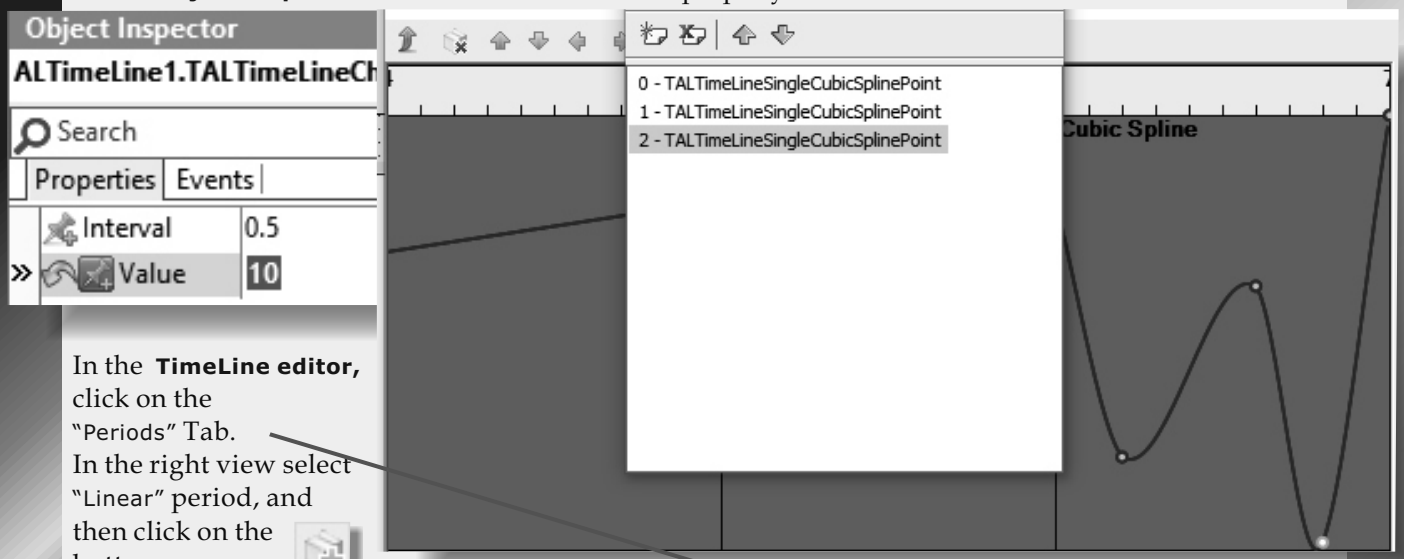




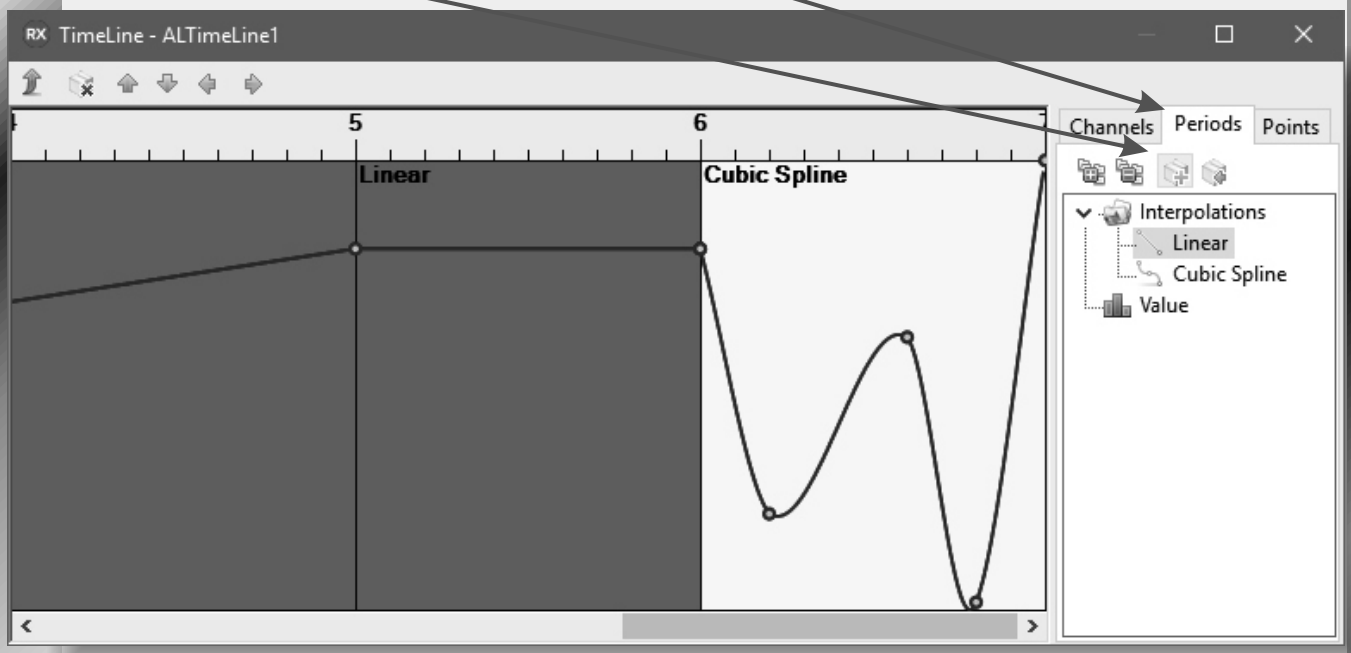
Select the second point in the collection editor.
In the **Object Inspector** set the value of the "Value" property to "40":



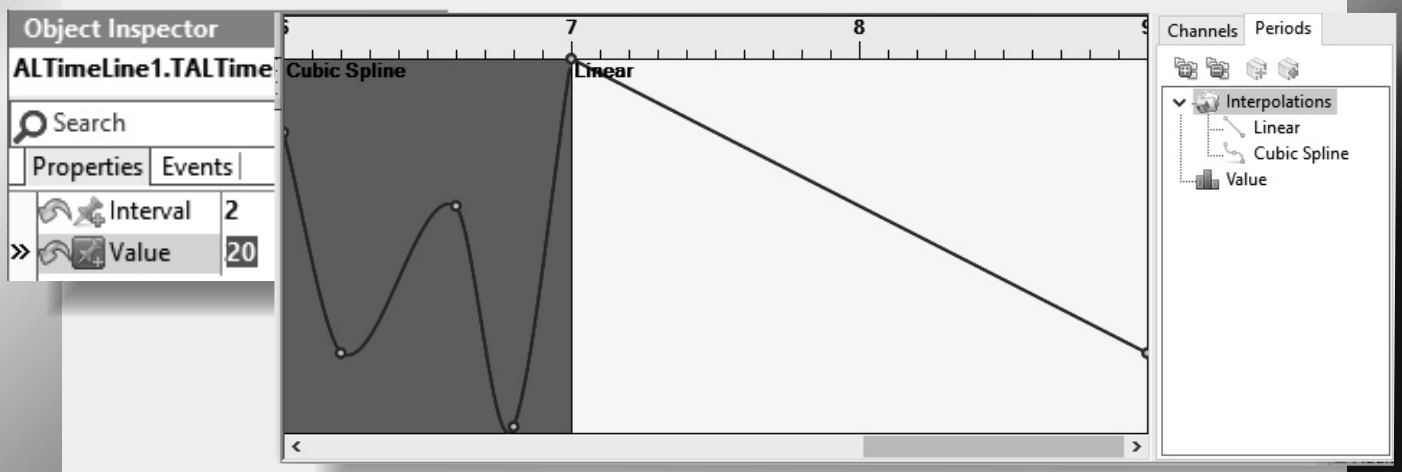
Select the third point in the collection editor.
In the **Object Inspector** set the value of the "Value" property to "10":



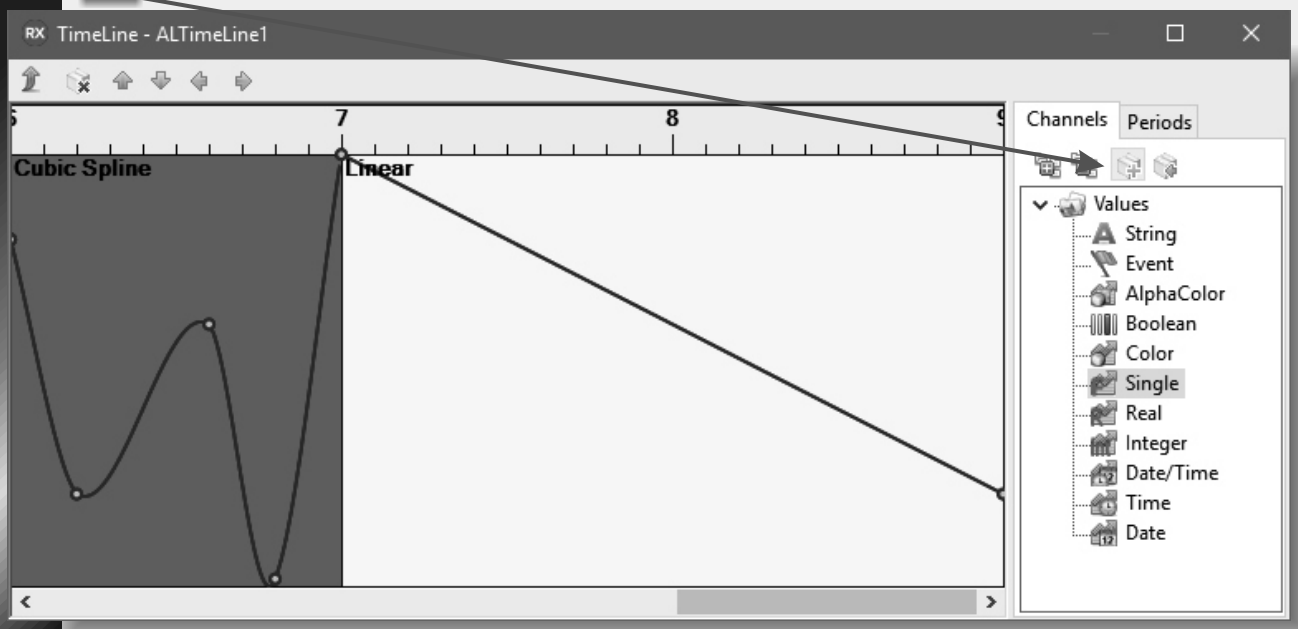
In the **TimeLine editor,** click on the "Periods" Tab.
In the right view select "Linear" period, and then click on the button:



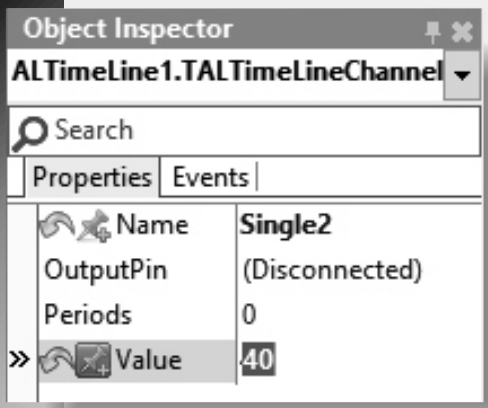
In the **Object Inspector** set the value of the "Interval" property to "2", and the "Value" property to "20":

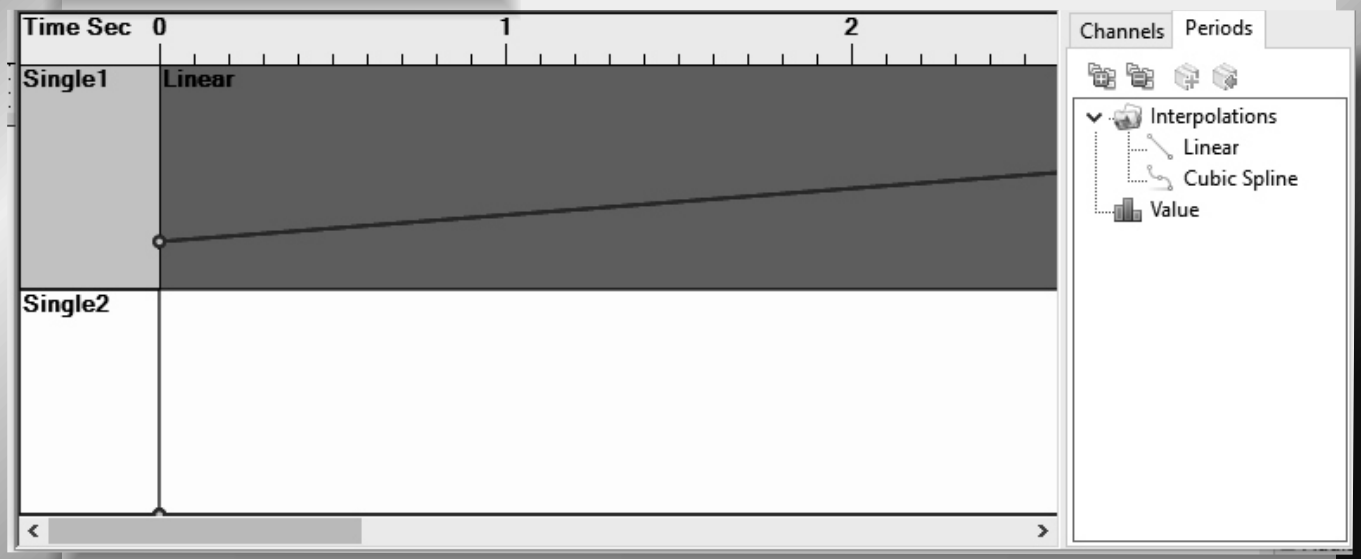


The first channel is done. Don't worry if you have missed a step. You can always return to edit the channel and its periods at a later point. Next we will add a second "Single" channel. In the right view of the TimeLine editor, click on the "Channels" tab. Select "Single" channel, and click on the button:

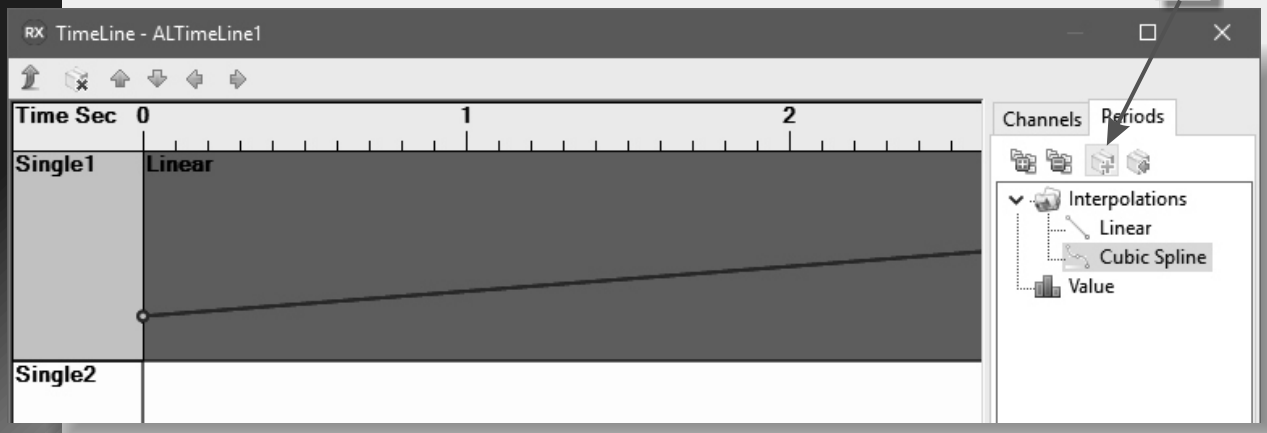


In the **Object Inspector** set the value of the "Value" property to "40":

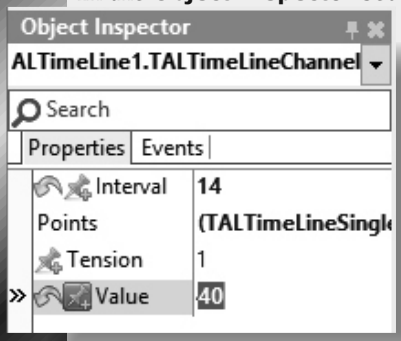


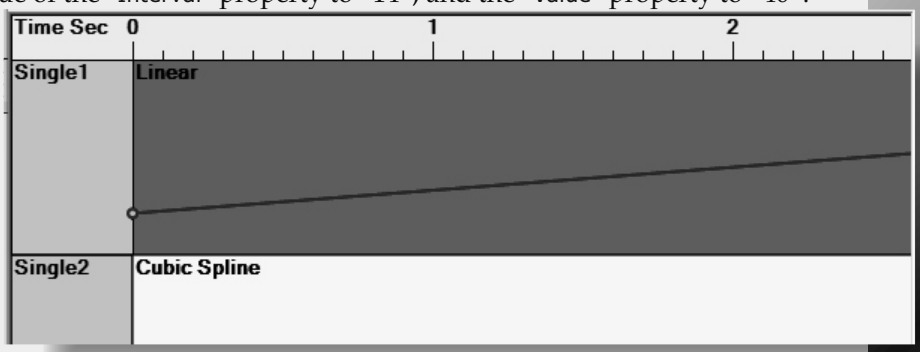


In the right view of the **TimeLine editor** select "Cubic Spline" period, and then click on the button:

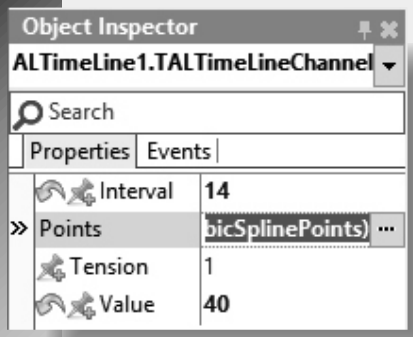


In the **Object Inspector** set the value of the "Interval" property to "14", and the "Value" property to "40":

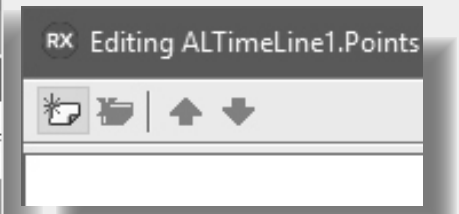




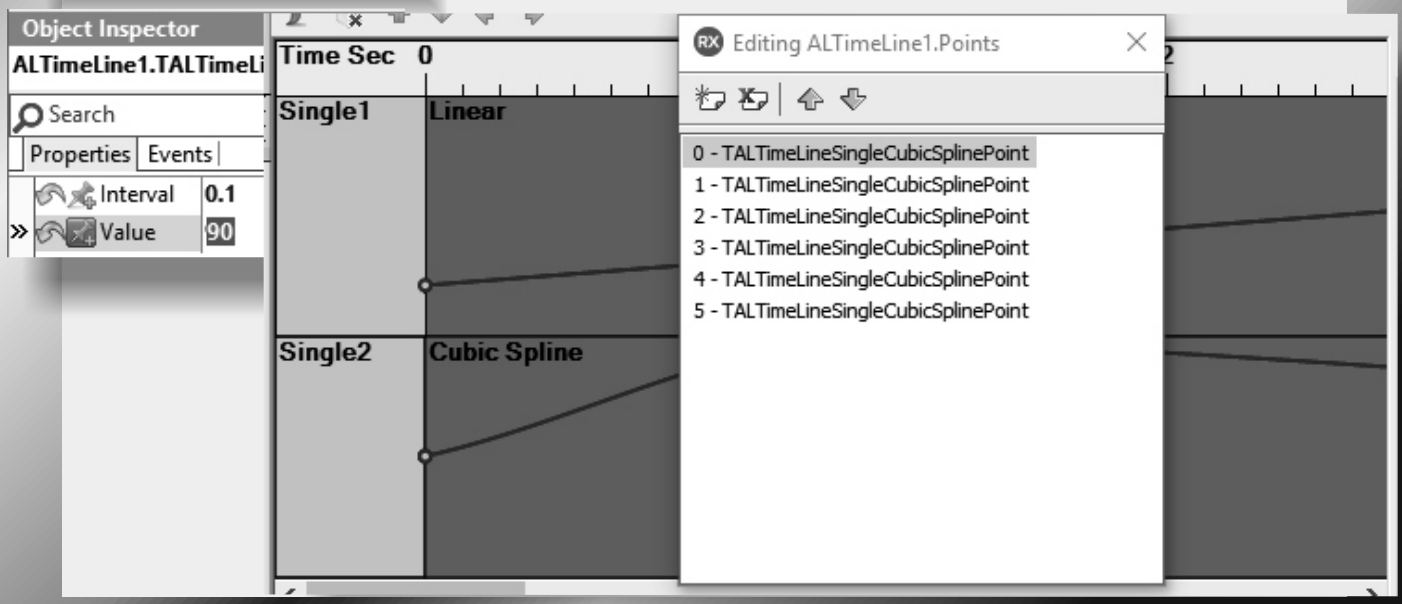
In the **Object Inspector** select the "Points" property, and click on the "..." button:



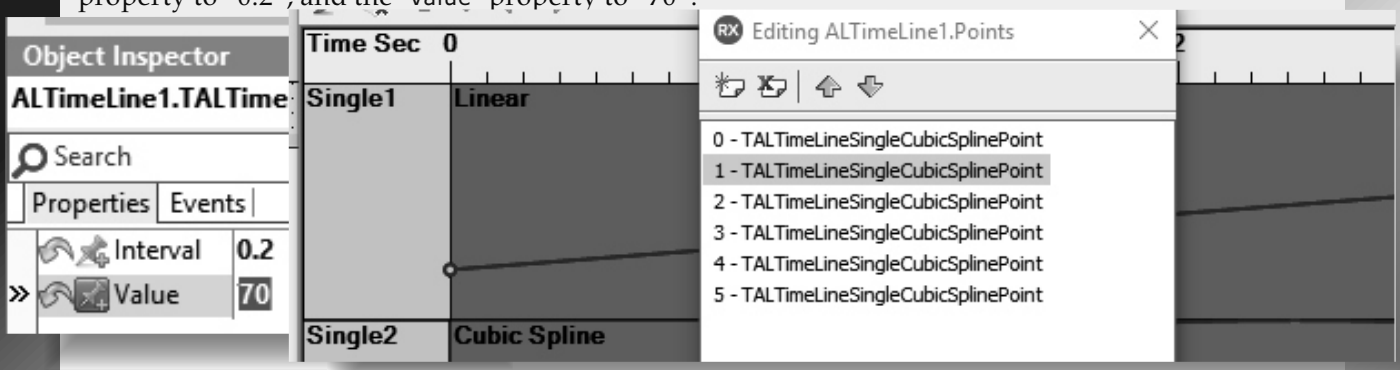
In the collection editor, click 6 times on the button to add 6 points:



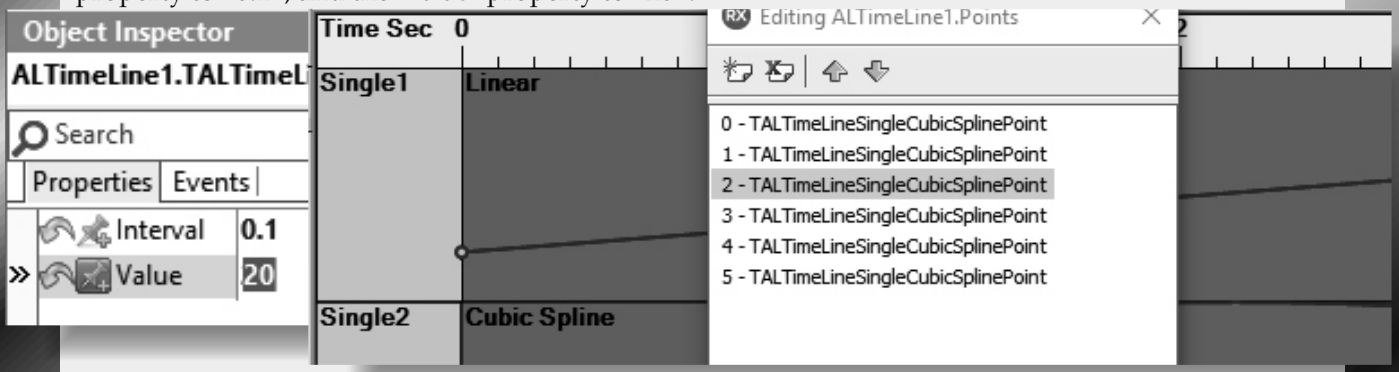
Select the first point.In the **Object Inspector** set the value of the "Interval" property to "0.1", and the "Value" property to "90":



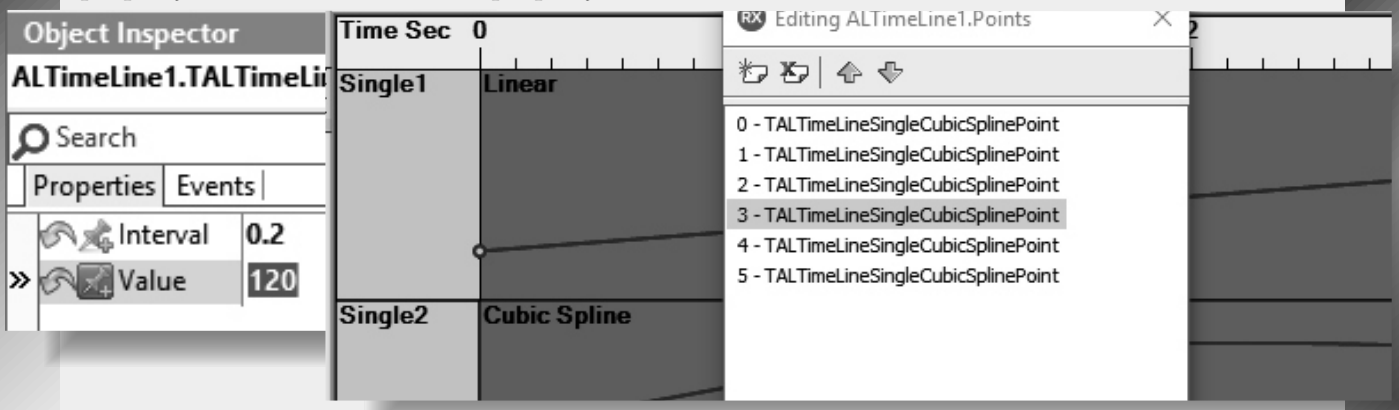
Select the second point in the collection editor. In the **Object Inspector** set the value of the "Interval" property to "0.2", and the "Value" property to "70":



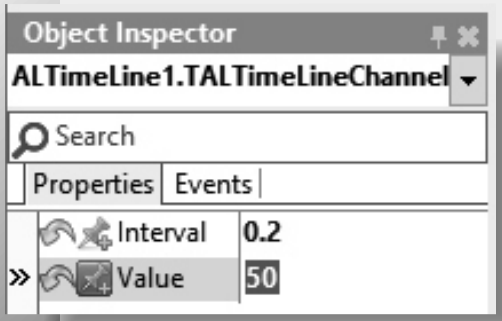
Select the third point in the **collection editor**. In the **Object Inspector** set the value of the "Interval" property to "0.1", and the "Value" property to "20":



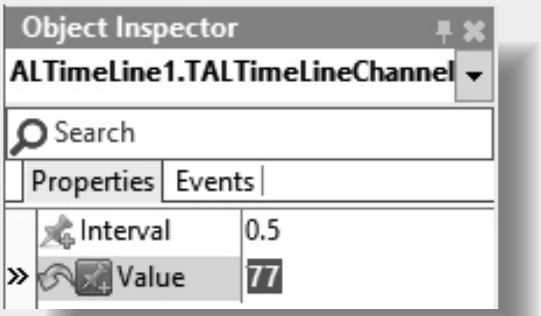
Select the 4th point in the collection editor. In the **Object Inspector** set the value of the "Interval" property to "0.2", and the "Value" property to "120":



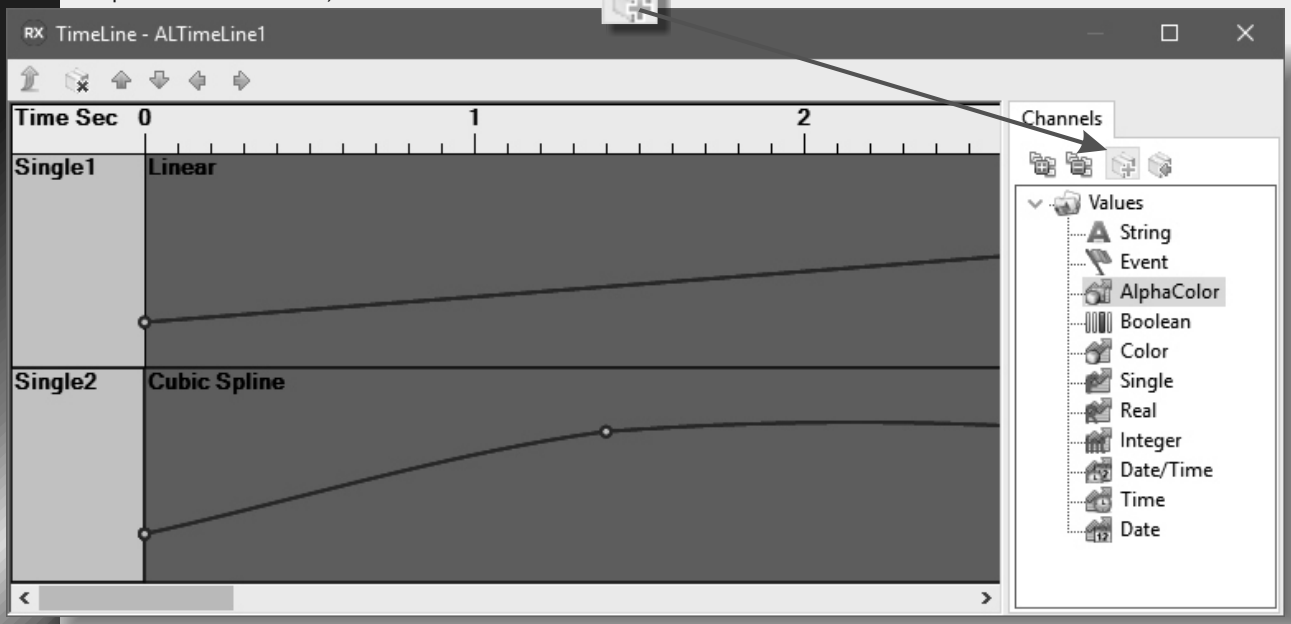
Select the 5th point in the collection editor. In the **Object Inspector** set the value of the "Interval" property to "0.2", and the "Value" property to "50":



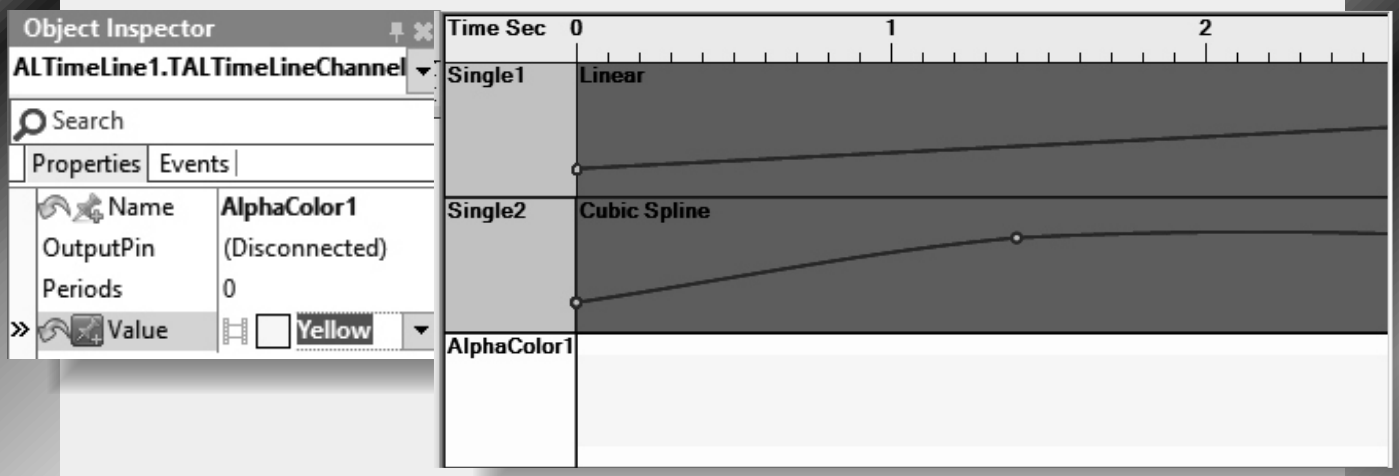
Select the 6th point in the collection editor. In the **Object Inspector** set the value of the "Value" property to "77":



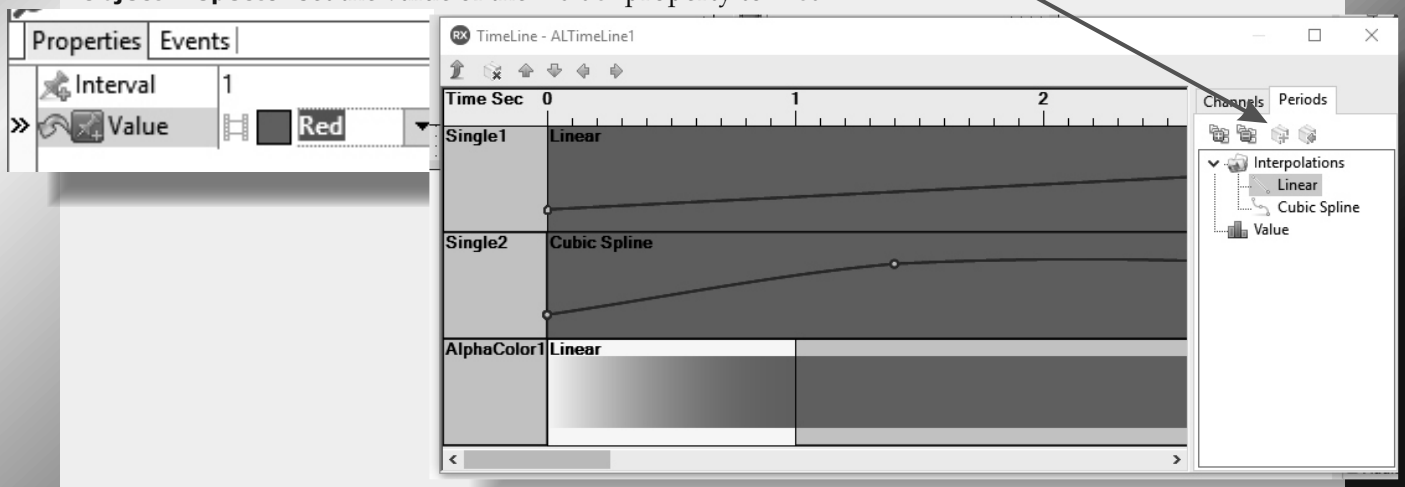
Close the **Points Collection Editor**. Now we will add an **Alpha Color channel** to control the Brush color of the rounded rectangle. In the right view of the TimeLine editor click on the "Channels" tab, select "AlphaColor" channel, and then click on the button:



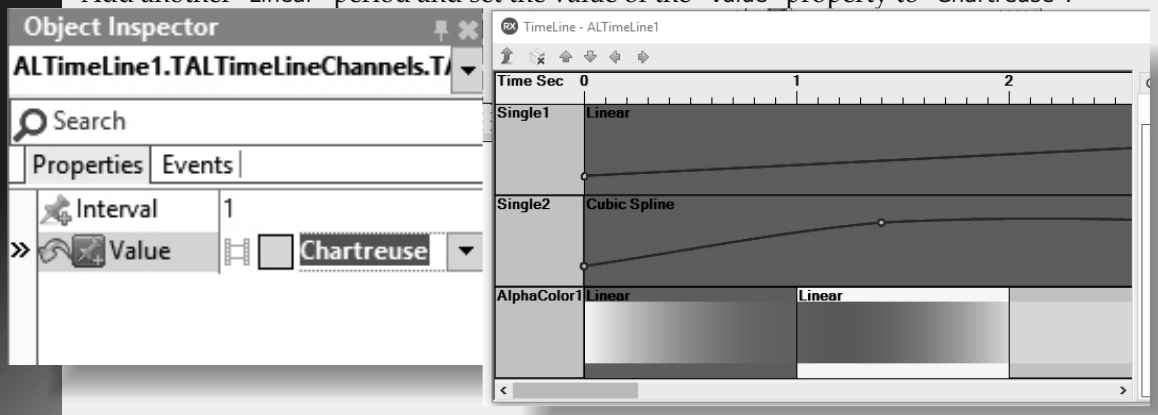
In the **Object Inspector** set the value of the "Value" property to "Yellow":



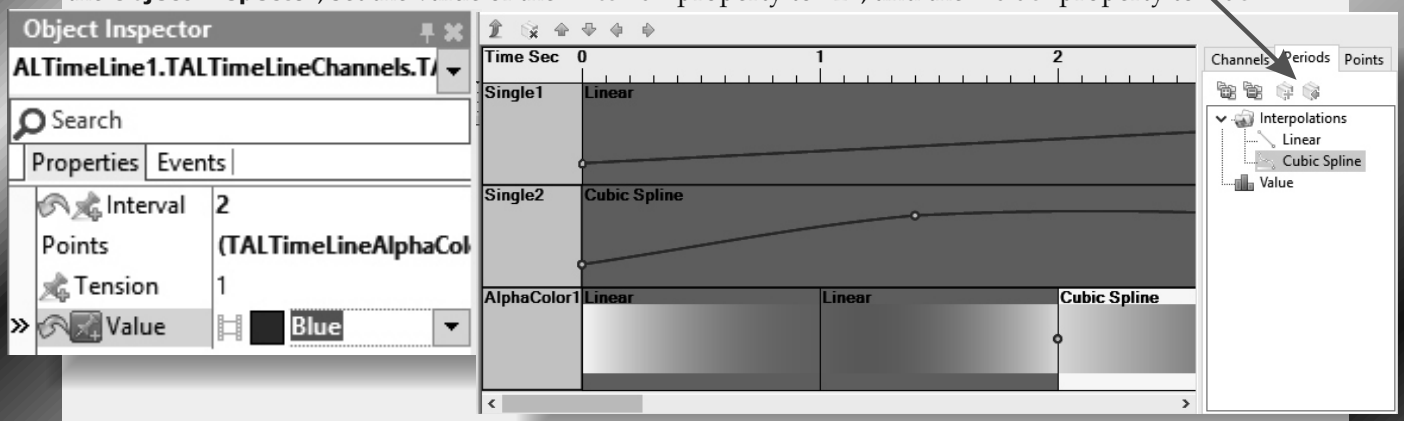
In the right view of the editor select "Linear" period, and then click on the button In the **Object Inspector** set the value of the "Value" property to "Red":



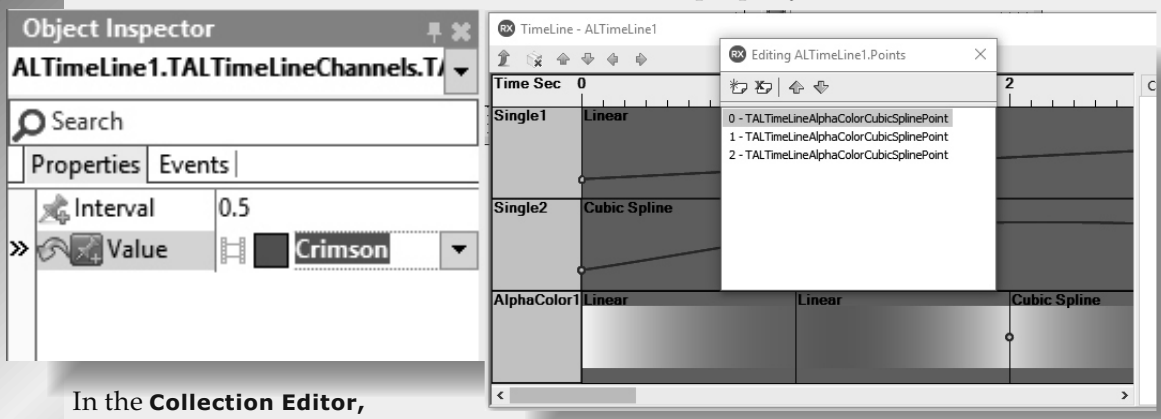
Add another "Linear" period and set the value of the "Value" property to "Chartreuse":



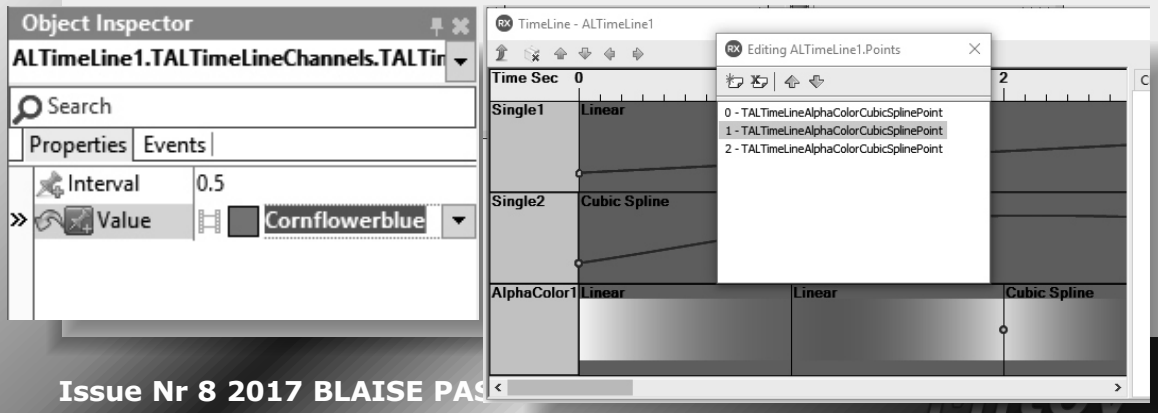
In the right view of the TimeLine editor select "Cubic Spline" period, and then click on the button. In the **Object Inspector**, set the value of the "Interval" property to "2", and the "Value" property to "Blue":



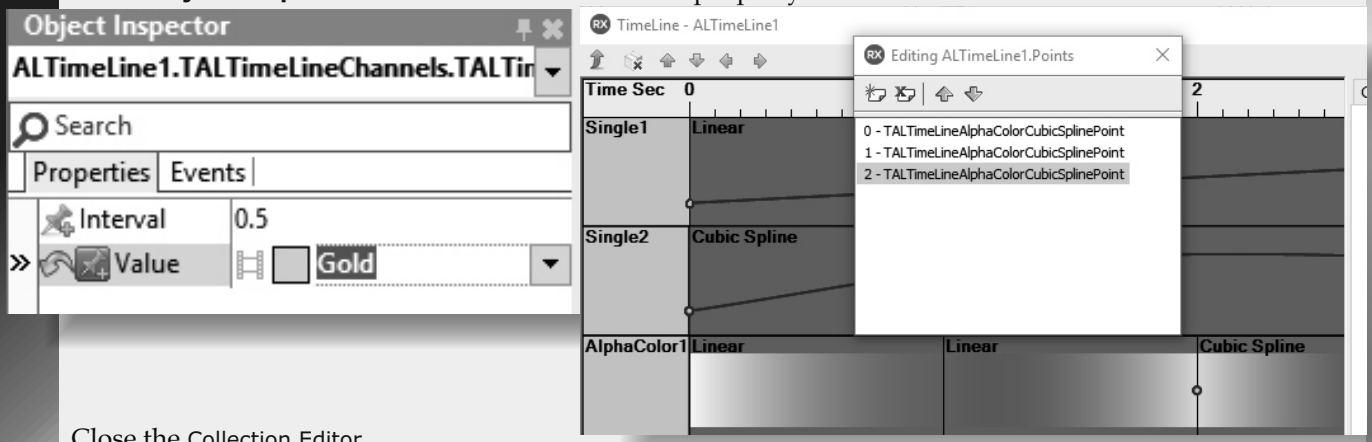
In the **Object Inspector** select the "Points" property, and click on the **"..."** button.
Add 3 Points to the **Cubic Spline Period**. In the **Collection Editor**, select the first point.
In the **Object Inspector** set the value of the "Value" property to "Crimson":



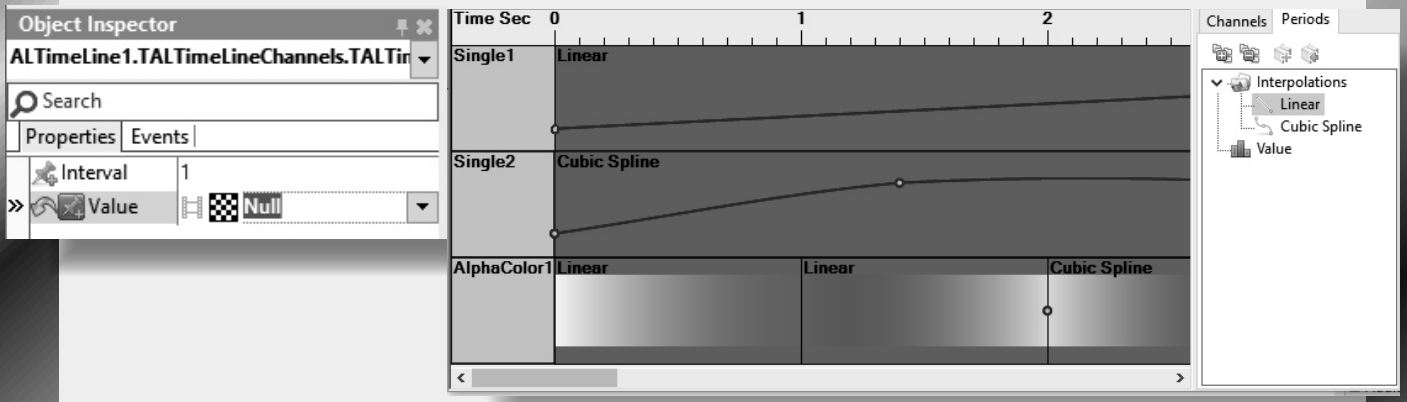
In the **Collection Editor,**
select the second point.
In the **Object Inspector** set the value of the "Value" property to "Cornflowerblue":



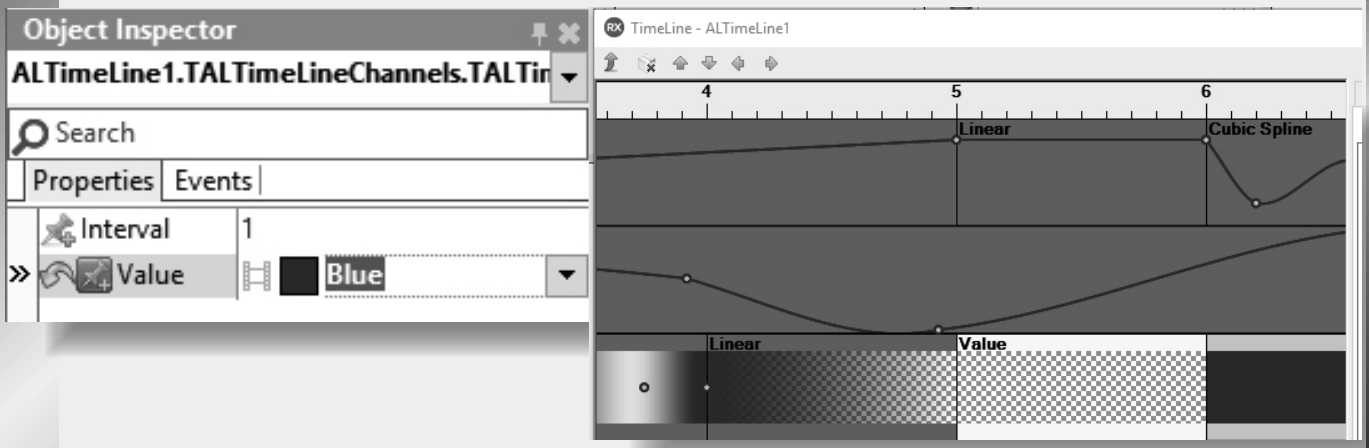
In the **Collection Editor,** select the third point.
In the **Object Inspector** set the value of the "Value" property to "Gold":



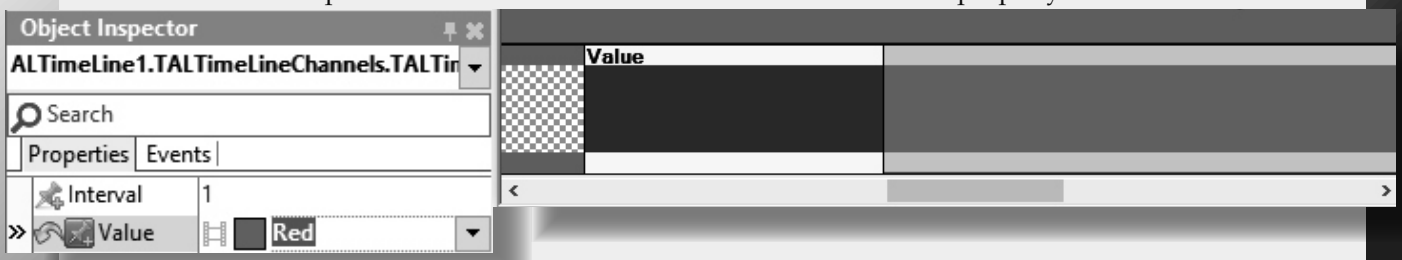
Close the Collection Editor.
Add another "Linear" period to the **TimeLine** and set the value of the "Value" property to "Null":



Add a "Value" period to the Channel and set the value of the "Value" property to "Blue":

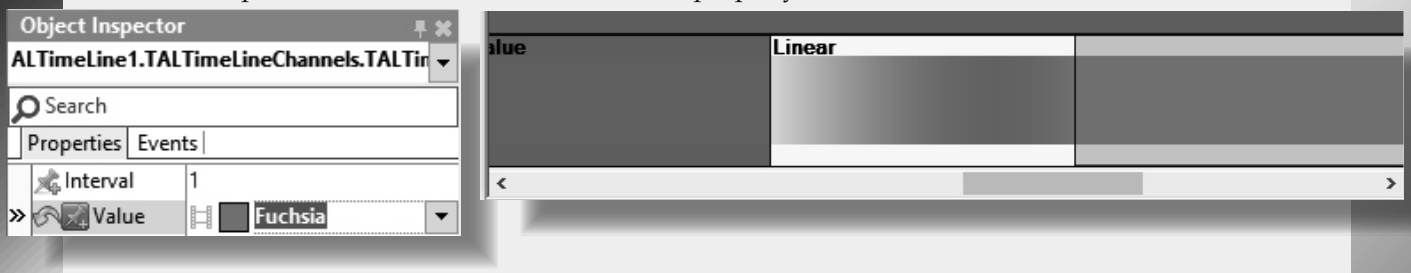


Add a second "Value" period to the Channel and set the value of the "Value" property to "Red":

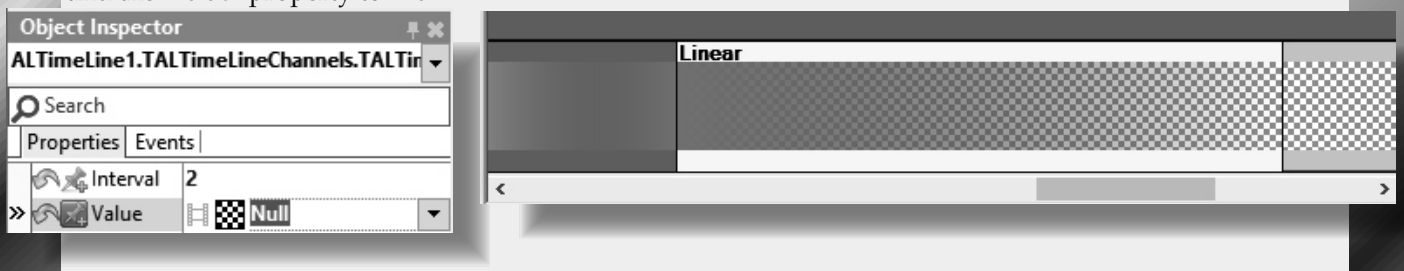


Add a third "Value" period to the Channel and set the value of the "Value" property to "Chartreuse":

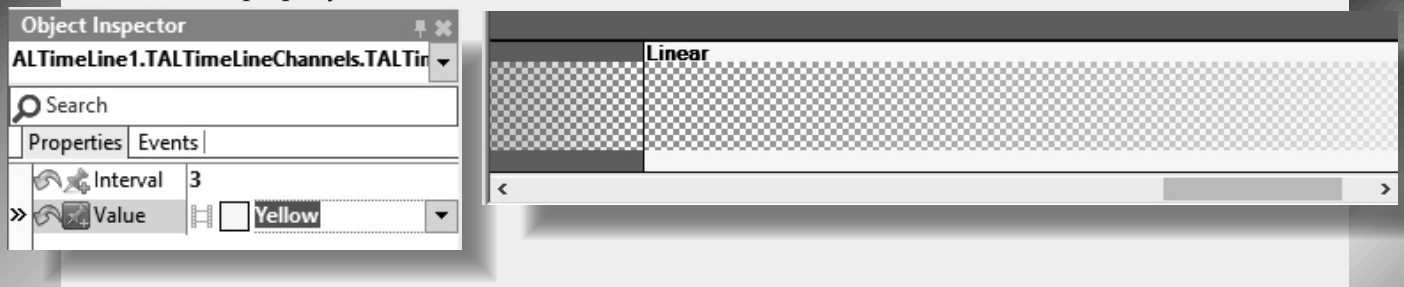
Add a "Linear" period and set the value of the "Value" property to "Fuchsia":



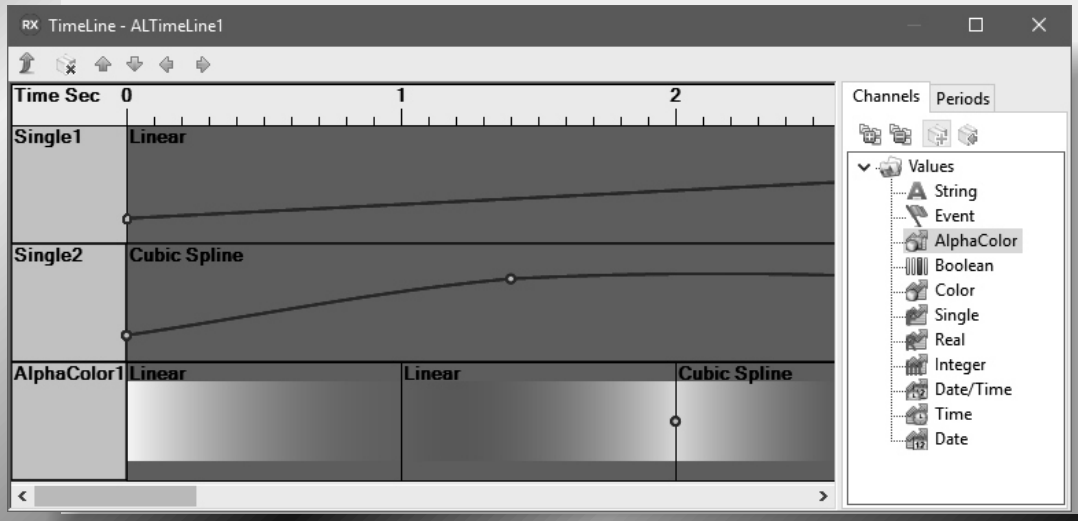
Add another "Linear" period and set the value of the "Interval" property to "2", and the "Value" property to "Null":



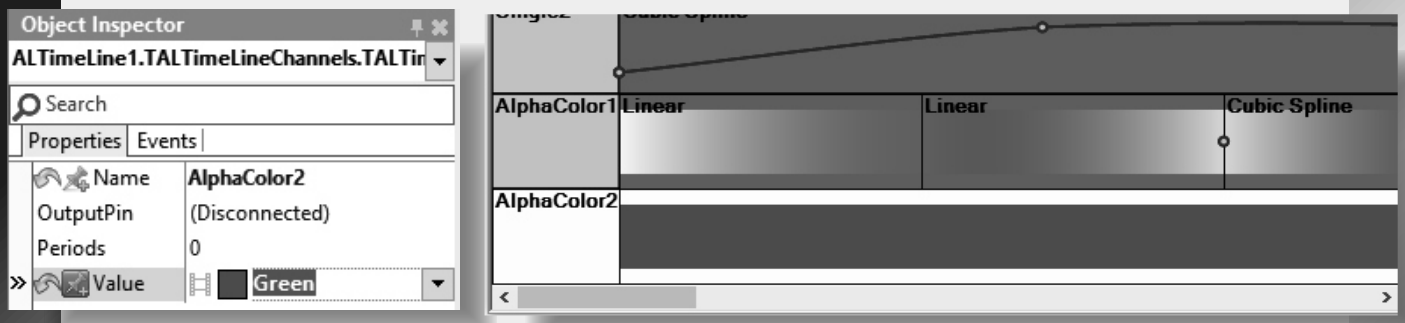
Add another "Linear" period and set the value of the "Interval" property to "3", and the "Value" property to "Yellow":



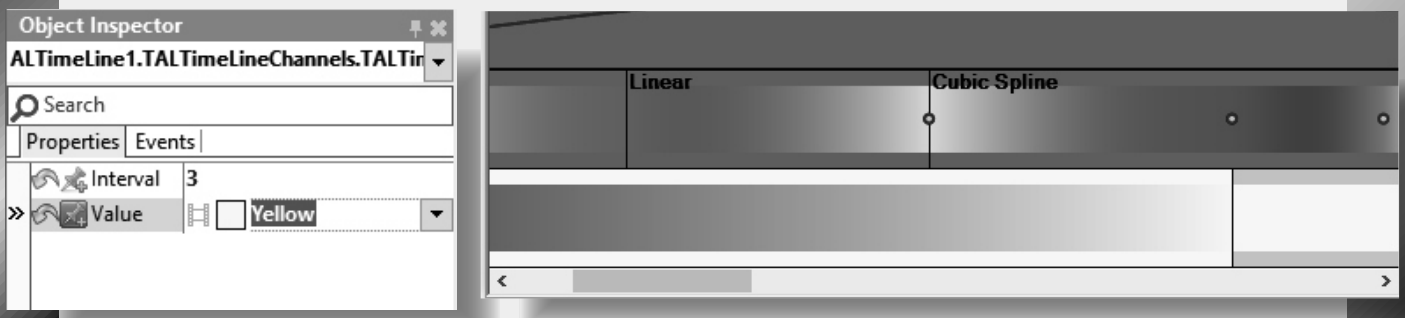
Next we will add an **Alpha Color channel** to control the "Pen.Brush" Color of the ellipse. In the right view of the **TimeLine editor** click on the "Channels" tab, select "AlphaColor" channel, and then click on the button:



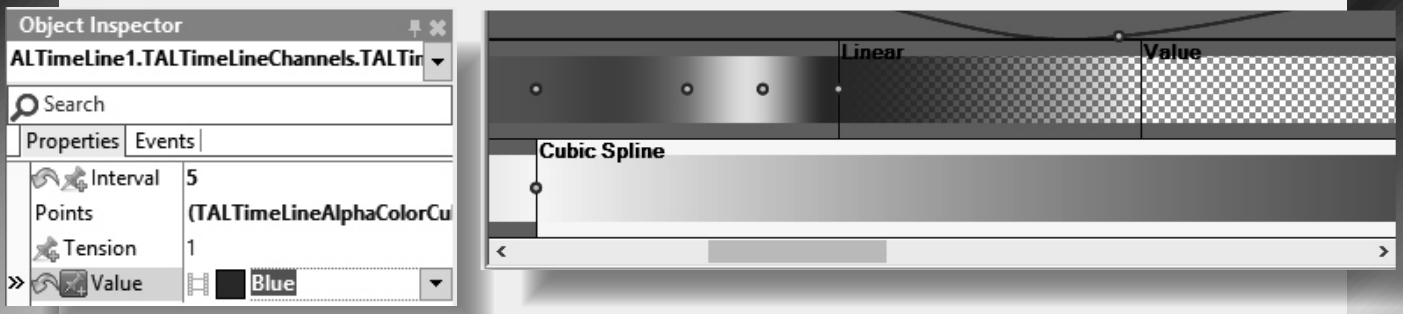
In the **Object Inspector** set the value of the "Value" property to "Green":



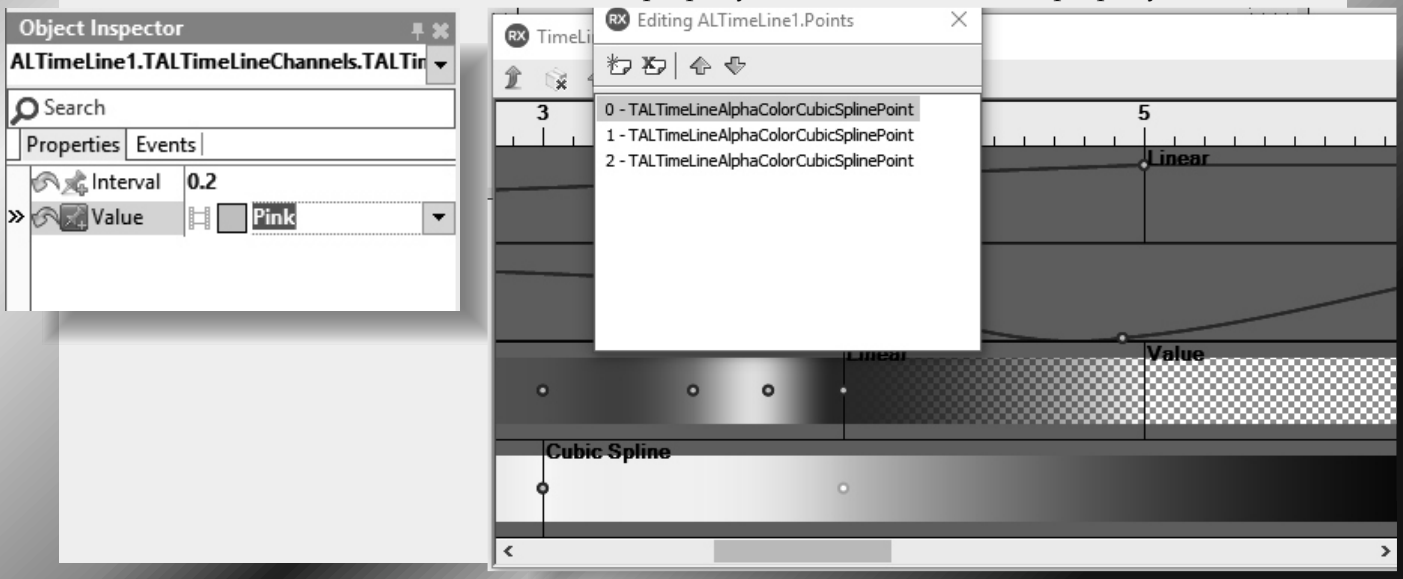
Add a "Linear" period and set the value of the "Interval" property to "3", and the "Value" property to "Yellow":



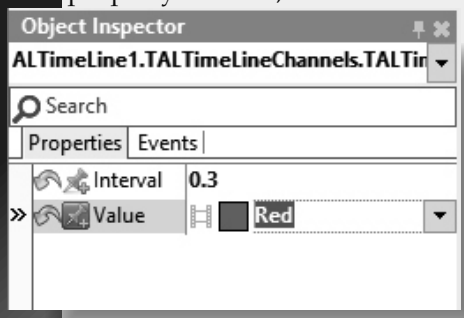
Add a "Cubic Spline" period, and set the value of the "Interval" property to "5", and the "Value" property to "Blue":

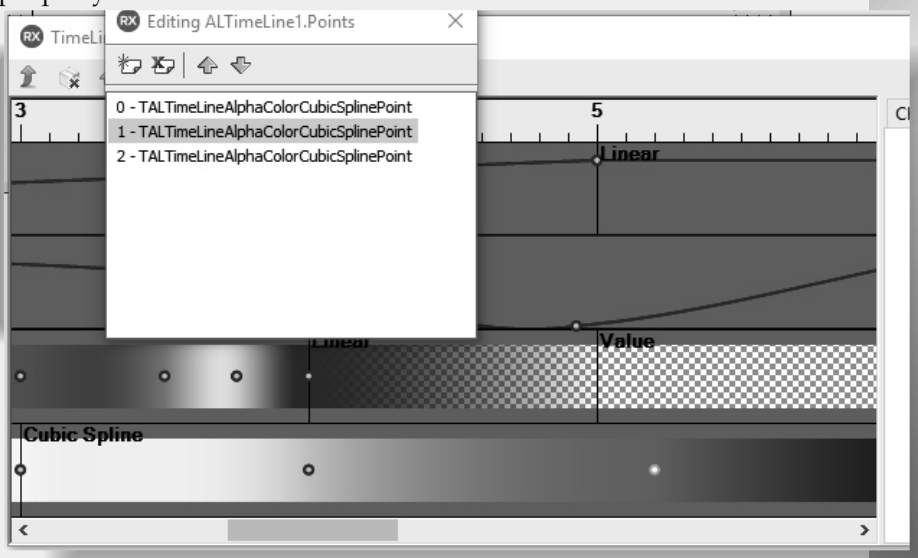


In the **collection editor**, click 3 times on the  button to add 3 points. Select the first point. In the **Object Inspector** set the value of the "Interval" property to "0.2", and the "Value" property to "Pink":

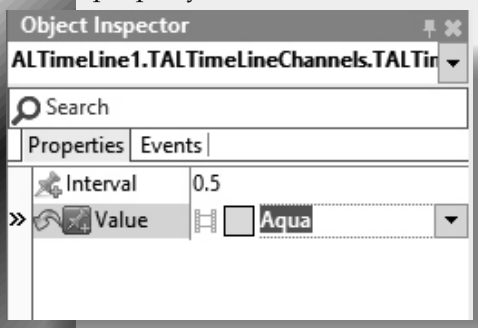


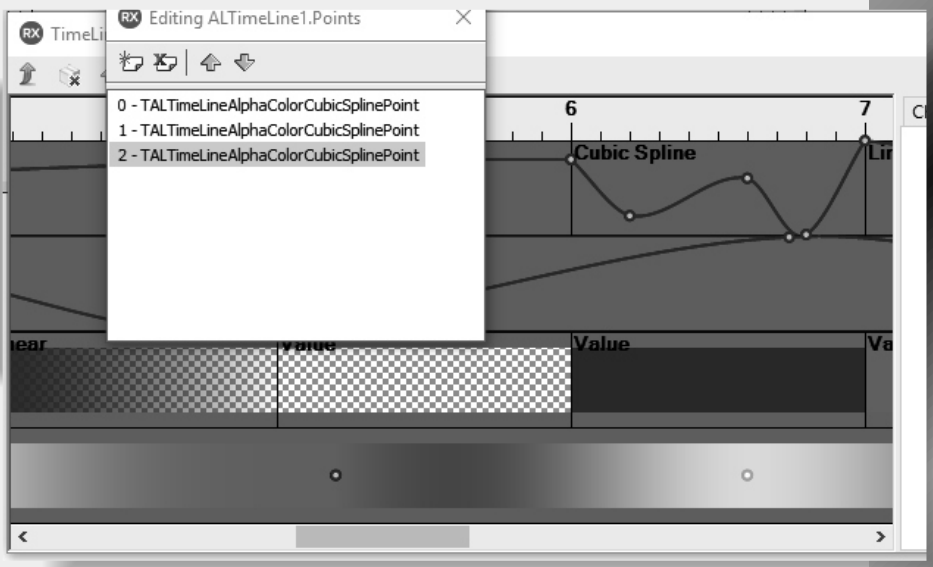
Select the second point in the collection editor. In the **Object Inspector** set the value of the "Interval" property to "0.3", and the "Value" property to "Red":



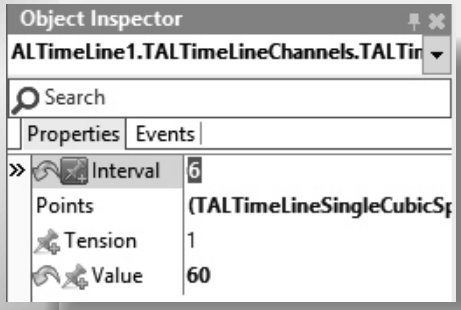


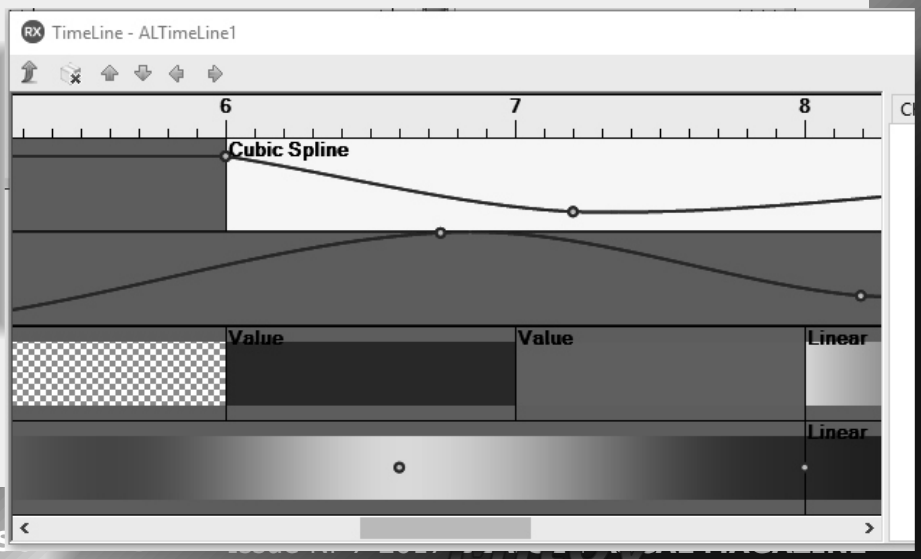
Select the third point in the **collection editor**. In the **Object Inspector** set the value of the "Value" property to "Aqua":



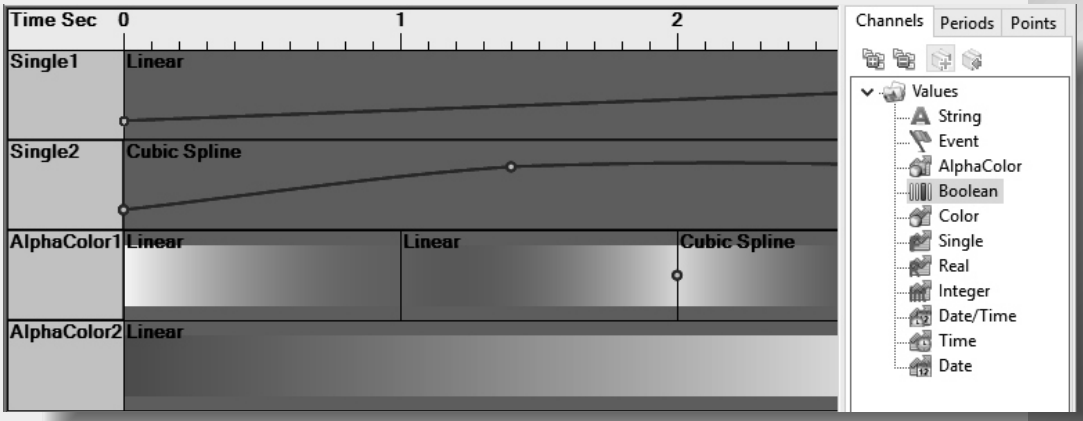


Close the **collection editor.**
Since we designed the first channel to be shorter than the rest, to make it more interesting we can select its "Cubic Spline" period and change its "Interval" property from "1" to "6". We can easily select the desired period on the **TimeLine.** Scroll the **TimeLine** until you see the "Cubic Spline" period on the first channel, and then click on the period. In the **Object Inspector** set the value of the "Interval" property to "6":

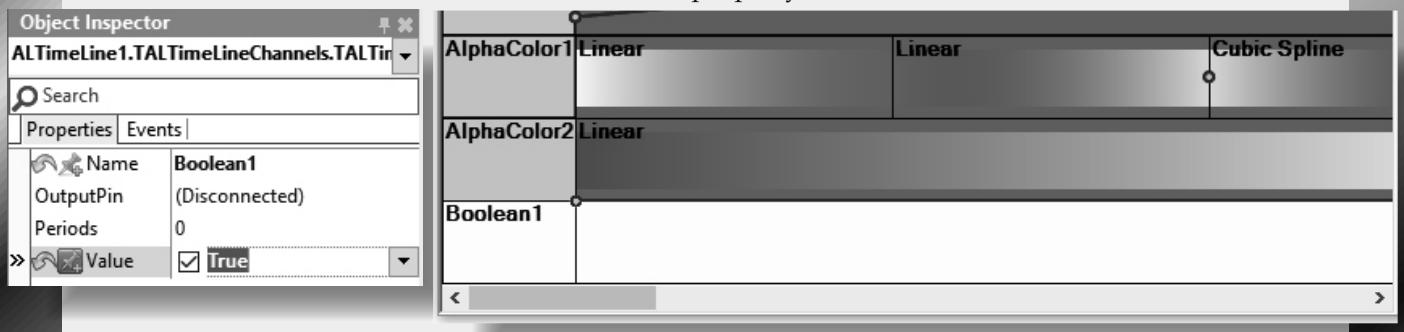




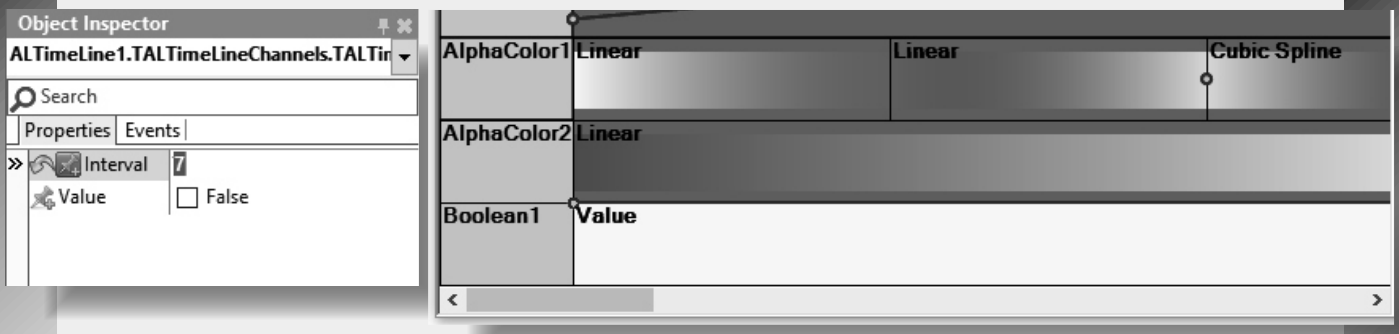
Now we will add a **Boolean channel** to Enable/Disable the Pen of the Ellipse.
In the right view of the **TimeLine editor,** click on the "Channels" tab. Select "Boolean" channel, and click on the button:



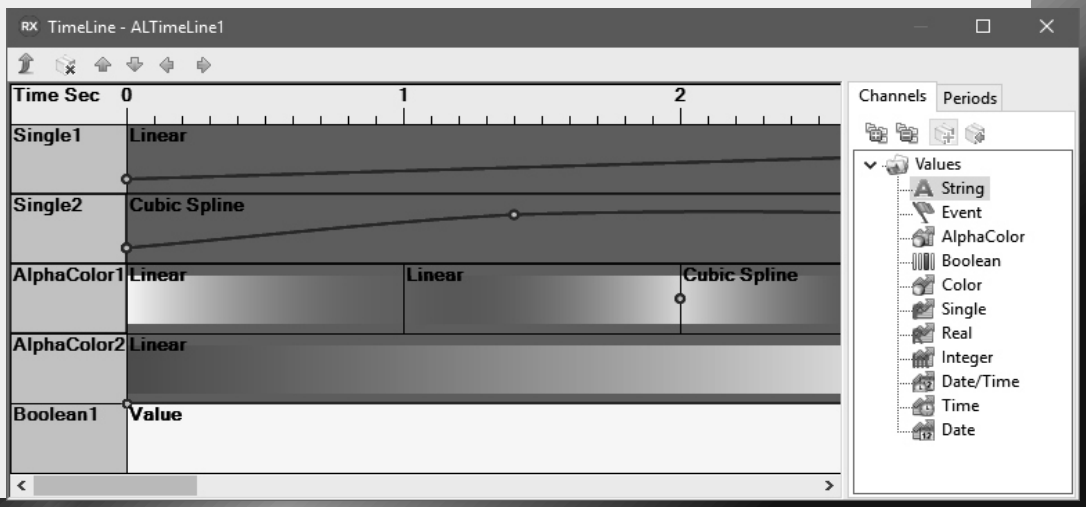
In the **Object Inspector** set the value of the "Value" property to "True":



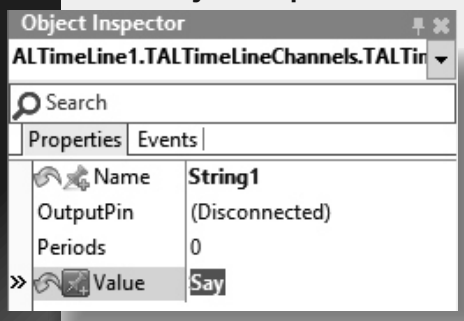
Add a "Value" period to the Channel and set the value of the "Interval" property to "7":



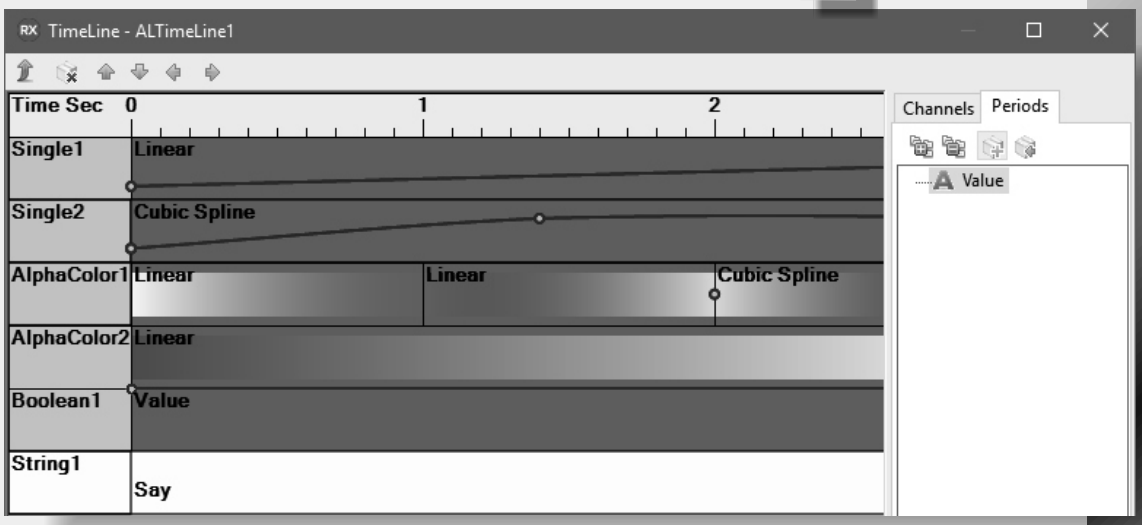
Finally we will add a **String channel** to change the text of the **Text Layer.**
In the right view of the **TimeLine** editor, click on the "Channels" tab. Select "String" channel, and click on the button



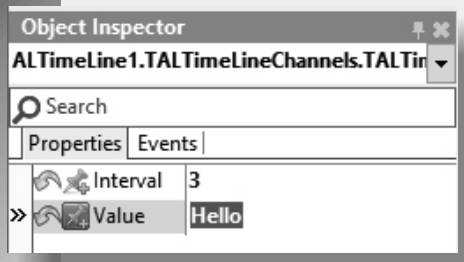
In the **Object Inspector** set the value of the "Value" property to "Say":

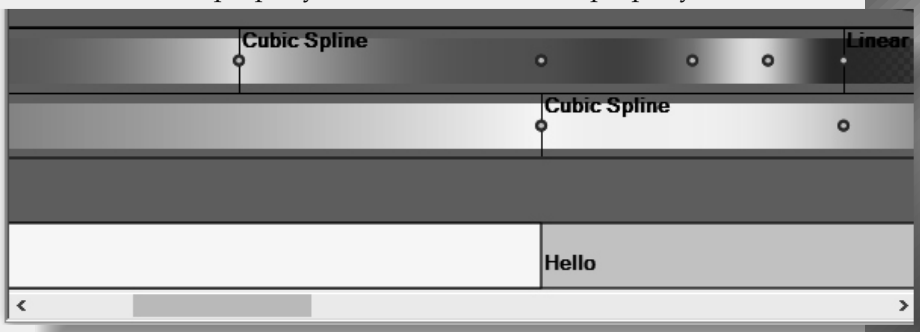


In the right view of the **TimeLine editor** select "Value" period, and then click on the  button

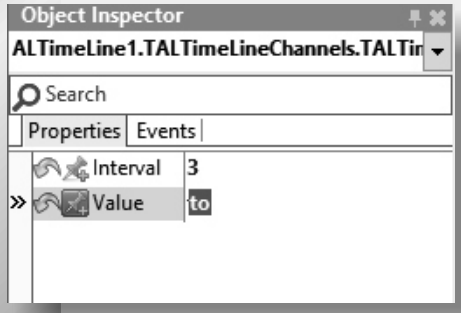


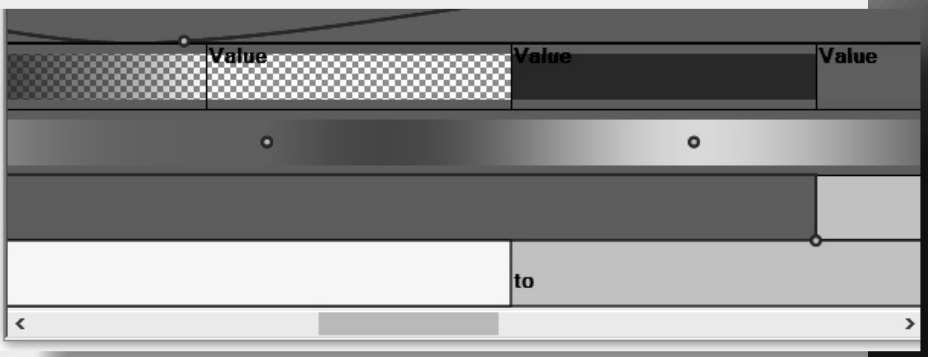
In the **Object Inspector** set the value of the "Interval" property to "3", and the "Value" property to "Hello":





Add second **"Value"** period to the channel.
In the **Object Inspector** set the value of the "Interval" property to "3", and the "Value" property to "to":





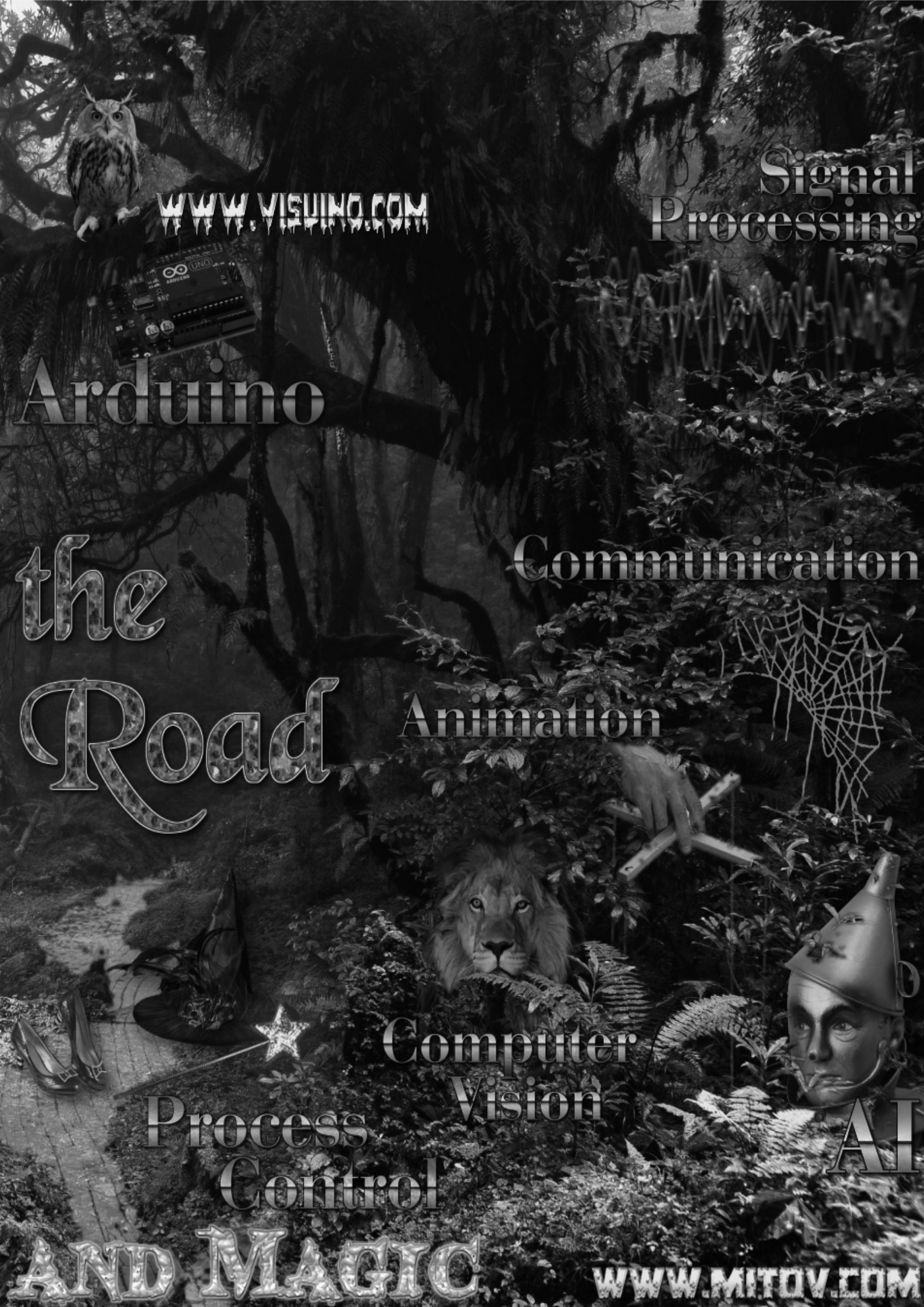
MIHOV SOFTWARE

Video

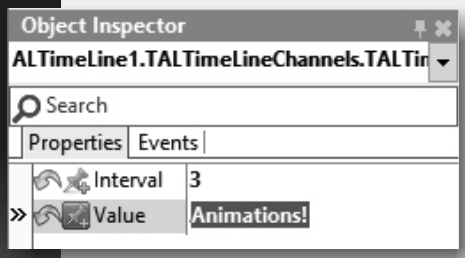
Follow Yellow Brick

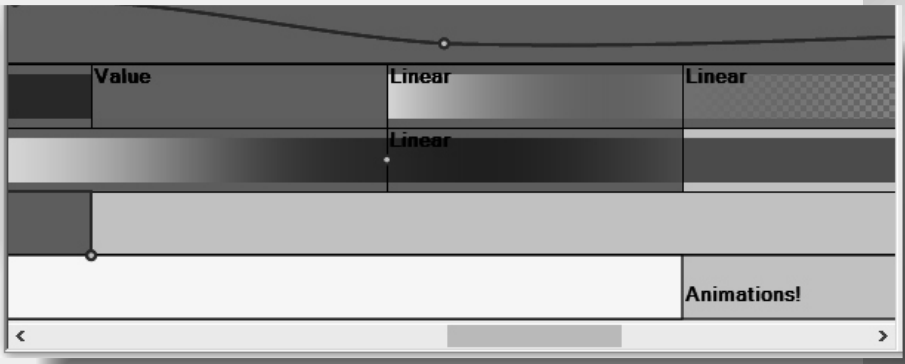
Audio

TO THE WORLD OF TECHNOLOGY

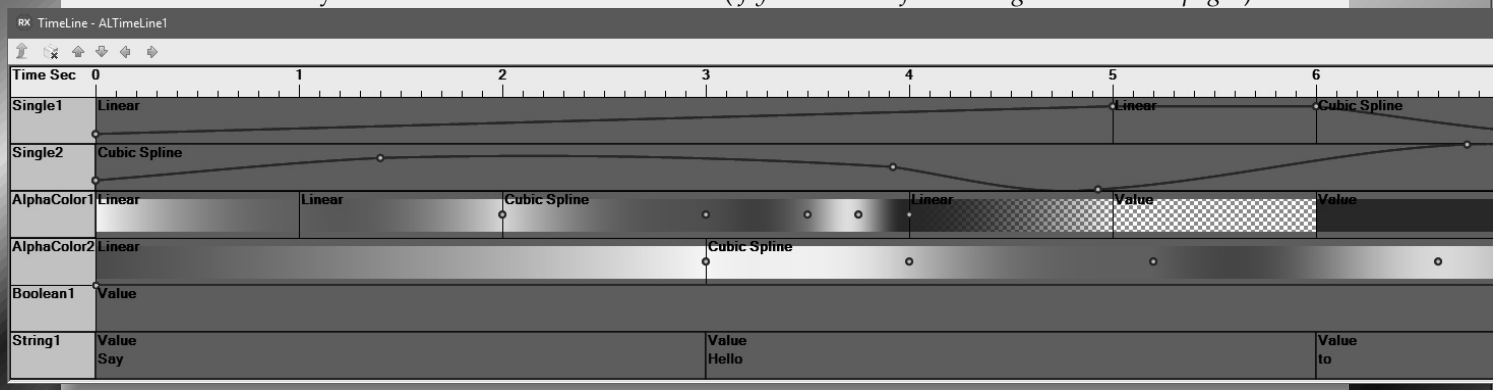

WWW.VISUINO.COM
Signal Processing
Arduino
Communication
the Road
Animation
Computer Vision
Process Control
AI
AND MAGIC
WWW.MITOV.COM

Add third "Value" period to the channel. In the **Object Inspector** set the value of the "Interval" property to "3", and the "Value" property to "Animations!":

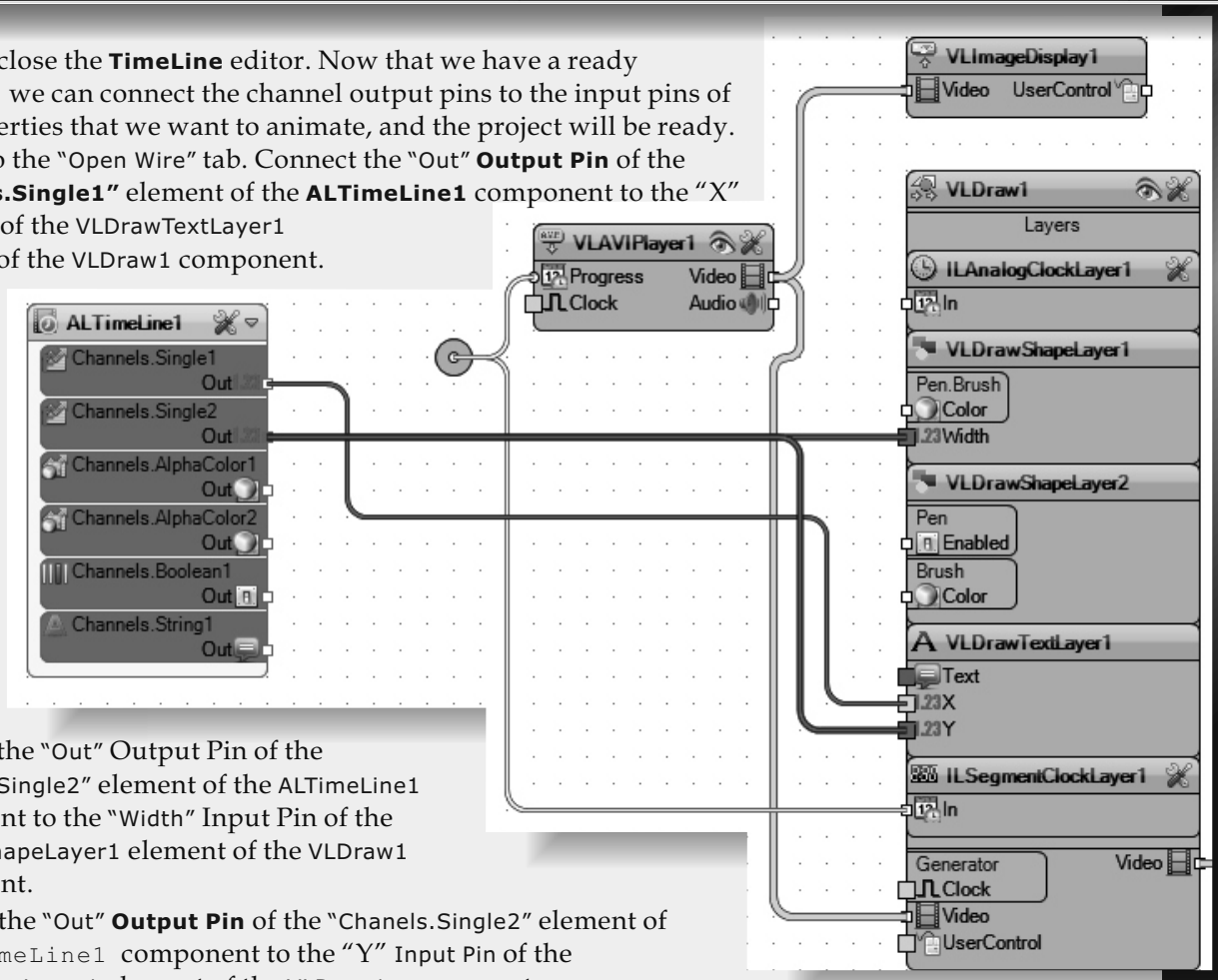




The **TimeLine** is ready. Here is how it should look like: (*if you use PDF for viewing switch to two pages*)



You can close the **TimeLine** editor. Now that we have a ready TimeLine, we can connect the channel output pins to the input pins of the properties that we want to animate, and the project will be ready. Switch to the "Open Wire" tab. Connect the "Out" **Output Pin** of the **"Chanels.Single1"** element of the **ALTimeLine1** component to the "X" Input Pin of the VLDrawTextLayer1 element of the VLDraw1 component.



Connect the "Out" Output Pin of the "Chanels.Single2" element of the ALTimeLine1 component to the "Width" Input Pin of the VLDrawShapeLayer1 element of the VLDraw1 component.

Connect the "Out" **Output Pin** of the "Chanels.Single2" element of the ALTimeLine1 component to the "Y" Input Pin of the VLDrawTextLayer1 element of the VLDraw1 component:

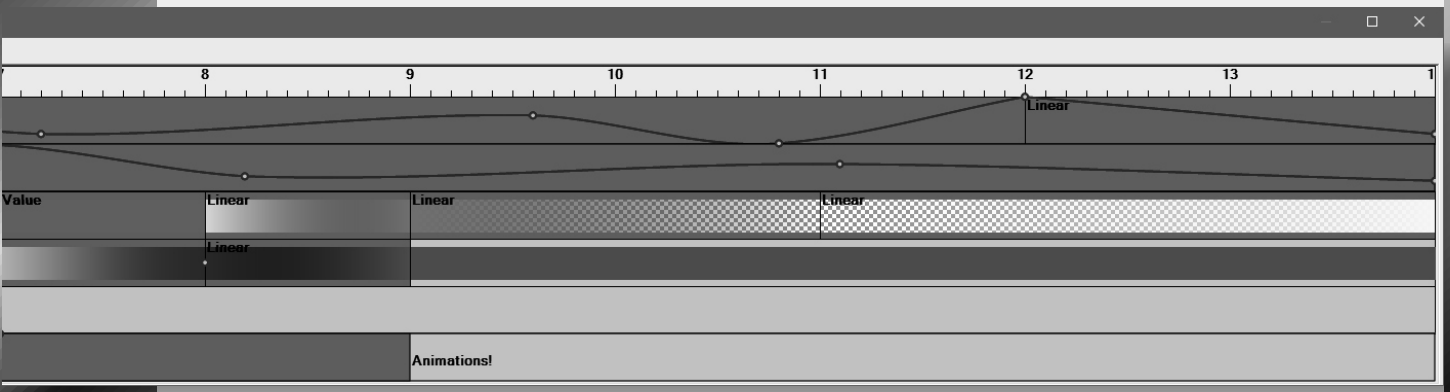
The "Chanels.Single2" will animate both the Y property of the text and the Width property of the **Rounded Rectangle.**

Connect the "Out" **Output Pin** of the "Chanels.AlphaColor1" element of the ALTimeLine1 component to the "Color" Input Pin of the "Brush" element of the VLDrawShapeLayer2 element of the VLDraw1 component.

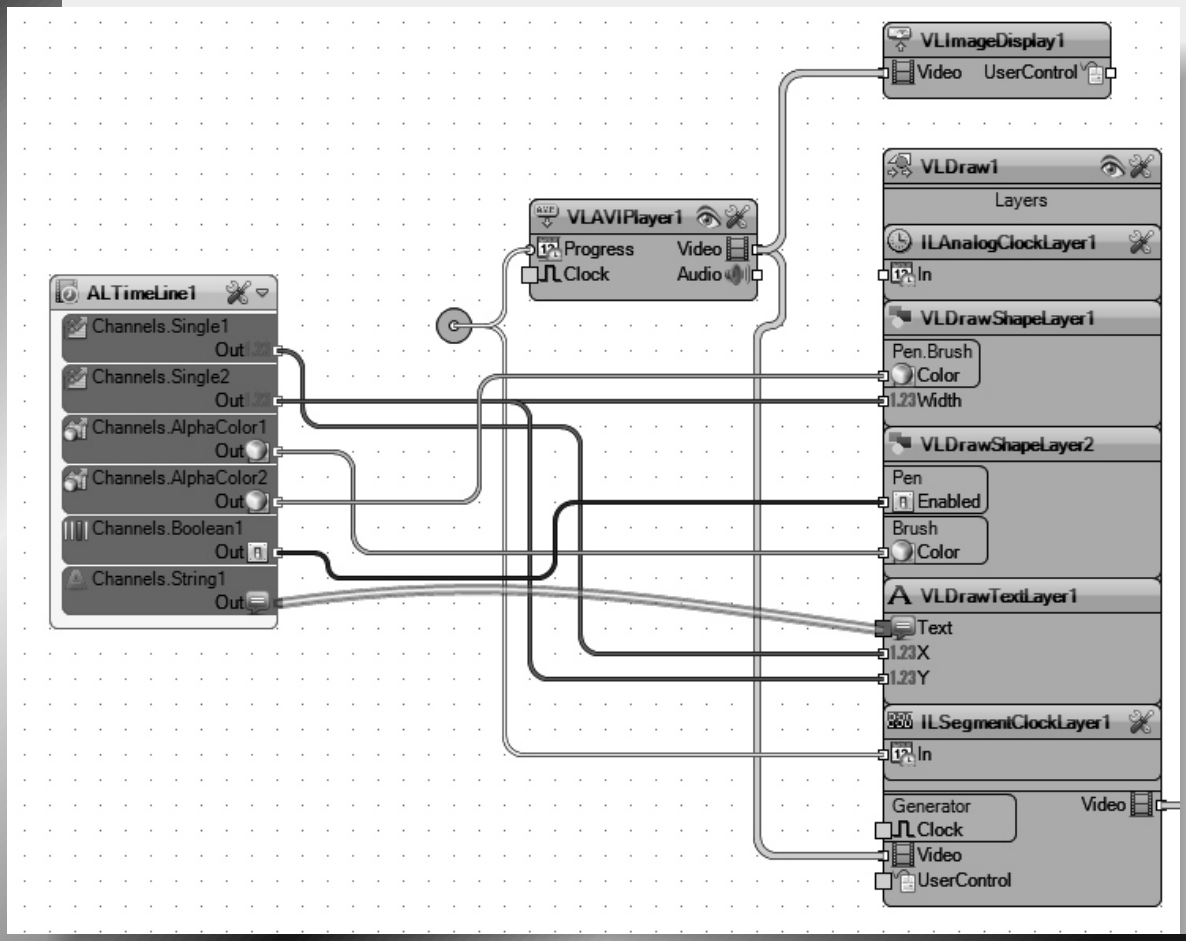
Connect the "Out" **Output Pin** of the "Chanels.AlphaColor2" element of the ALTimeLine1 component to the "Color" Input Pin of the "Pen.Brush" element of the VLDrawShapeLayer1 element of the VLDraw1 component.

Connect the "Out" **Output Pin** of the "Chanels.Boolean1" element of the ALTimeLine1 component to the "Enabled" **Input Pin** of the "Pen" element of the VLDrawShapeLayer2 element of the VLDraw1 component.

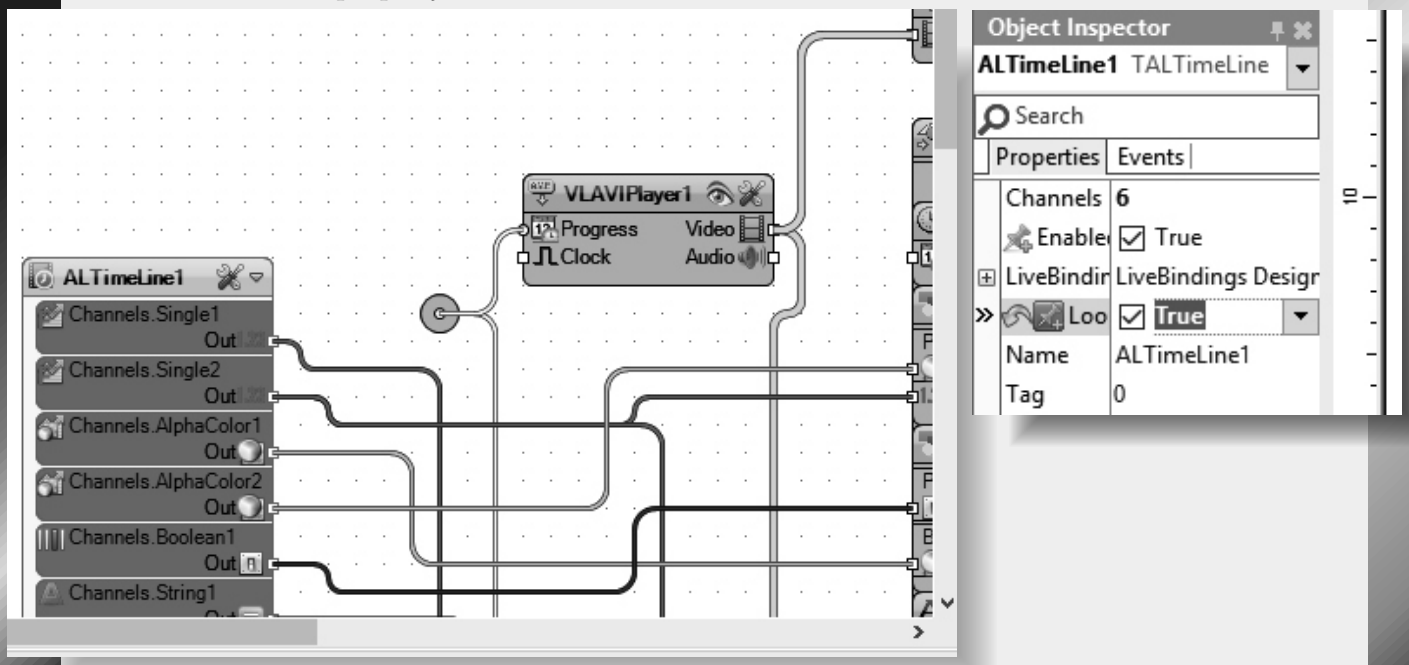
Connect the "Out" **Output Pin** of the "Chanels.String1" element of the ALTimeLine1 component to the "Text" Input Pin of the VLDrawTextLayer1 element of the VLDraw1 component:



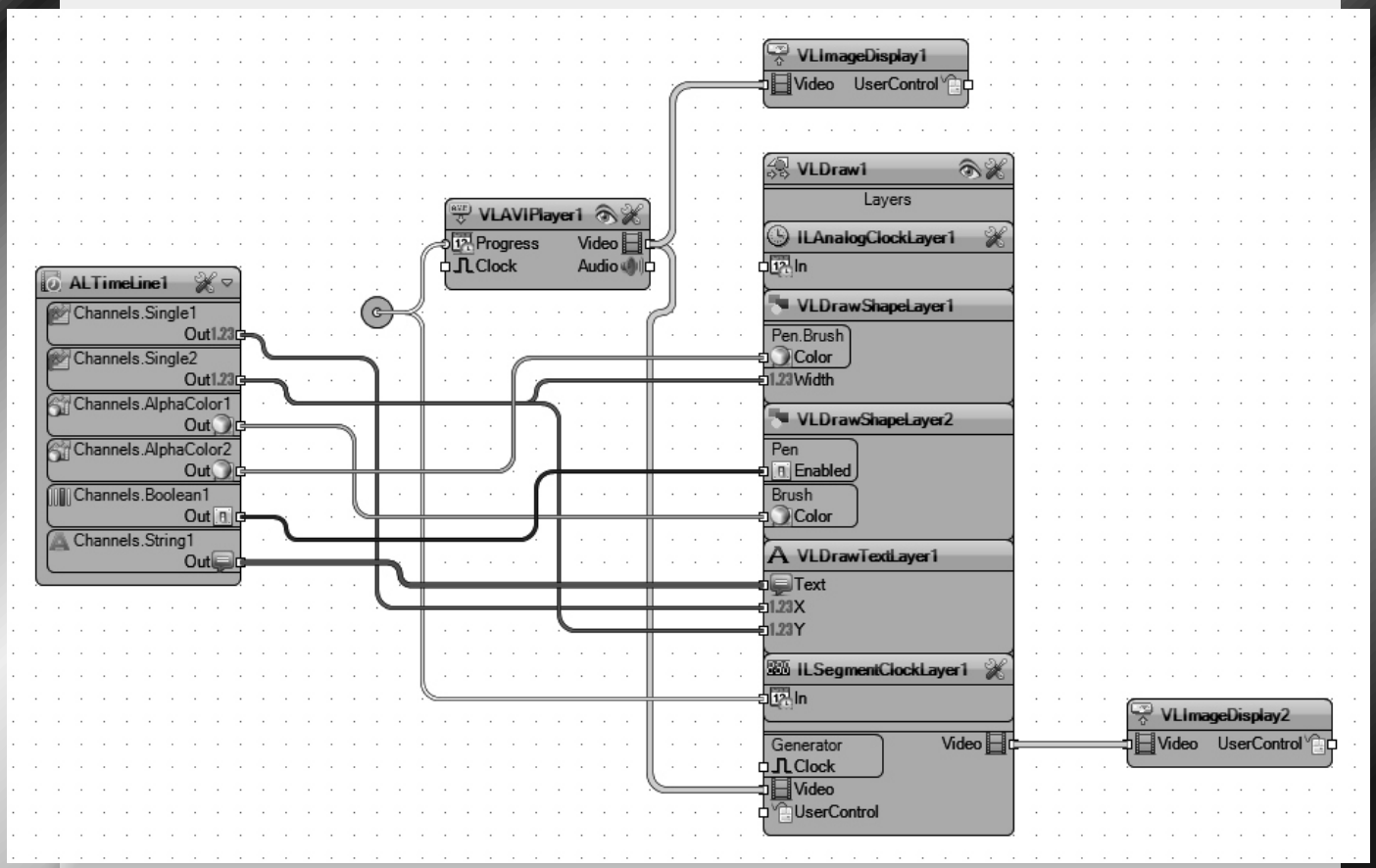
(*if you use PDF for viewing switch to two pages*)



By default the **TimeLine** will execute only once, and will stop. To have it automatically restart, we can set the "Loop" property to "True":



Here is the complete **OpenWire diagram** of the project:



COMPILE AND RUN THE APPLICATION.
You should see the **Rounded Rectangle's** border changing color over time, the **Rounded Rectangle**'s "Width" changing, the **Circle's background** changing color, the **Circle's border** appearing and disappearing, and the **text changing and moving around, while still being on fire**:

Congratulations!
You have learned how to animate video layers with TimeLine animation!

CONCLUSION
In this and the previvous article, I showed you how you can add video layers to your video, apply effects to the layers, and how to use Animation TimeLine and Visual Live Binding to animate the layers.
In the following Articles I will show you how you can use other video sources such as directly connected to the computer Cameras, TV Tuners, Remote IP Cameras, Internet Video Streams, Images, or even generating your own video from code.
I will also show you how you can Analyze the video, Mix different video sources, save the video, send it to other devices, or broadcast it over the internet.

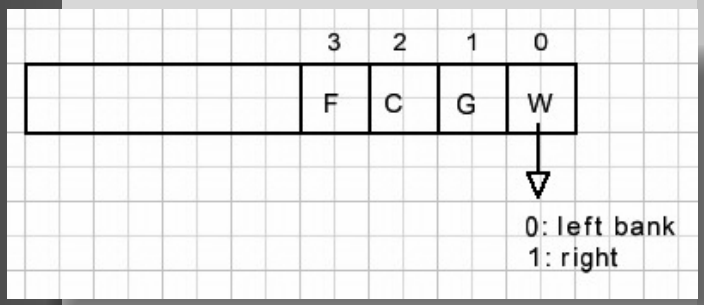
RIVERCROSS PROBLEMS PAGE 1/4
BY DAVID DIRKSE

INTRODUCTION
This article describes two very similar puzzles and their solution by means of Delphi programs. The two projects are called rivercross1 and rivercross2. Both are classical and well known problems.
(1) Is the problem of a farmer which has to row a cabbage, a goat and a wolf to the other bank of the river. The boat only allows to carry one item at the time.
The farmer may not leave the goat and cabbage unattended: the goat would eat the cabbage. Also the wolf and the goat may not be left unattended, for obvious reasons.
(2) Is the jealous husbands problem.
Three couples have to cross a river. The boat has space for 2 people only.
Both husbands and wives may row the boat. Problem is that the jealous husbands do not tolerate that their wife is in company of another man when they are at an opposite bank, even not when the other mans wife is present

Focus is on the search algorithm.
No efforts are taken to draw nice graphics. Results are simply presented as a list. Items are indicated by characters.
Below first follows a description of the riverside1 program.
This problem is rather simple.

The riverside2 problem is more difficult however the program is very similar.
Only the differences with rivercross1 will be explained.

RIVERCROSS 1
Programming is writing procedures and functions that operate on data structures.
What data is needed?
The items to move are the farmer (F) which rows the boat, the cabbage (C), the goat (G) and the wolf (W).
Their position is at the left- or the right bank of the river.
So, we need one bit per item 0: left bank 1: right bank.
To switch banks (row to opposite bank) requires the logical operation xor 1
Variable situation holds the 4 bits that indicate the place of each item:



There are 4 items so situations range from 0 to 15 {2^4 – 1 = (1 shl 4) - 1}

When moving between banks, the boat may be occupied by

1. Farmer + Cabbage
2. Farmer + Goat
3. Farmer + Wolf
4. Farmer alone

`ActionCode[0..4]` array holds (binary) values
0: 0000 0000 no action
1: 0000 1100 move Farmer + Cabbage
2: 0000 1010 move Farmer + Goat
3: 0000 1001 move Farmer + Wolf
4: 0000 1000 move Farmer alone
We have to remember which actions were taken.
The crossList[1..20] array holds action numbers 0..4.

Variable crossNr is the pass of the boat: 1,2,3....
Odd passes are from the left to the right bank, even passes are returns. There is no need to have a special variable holding the boat direction.

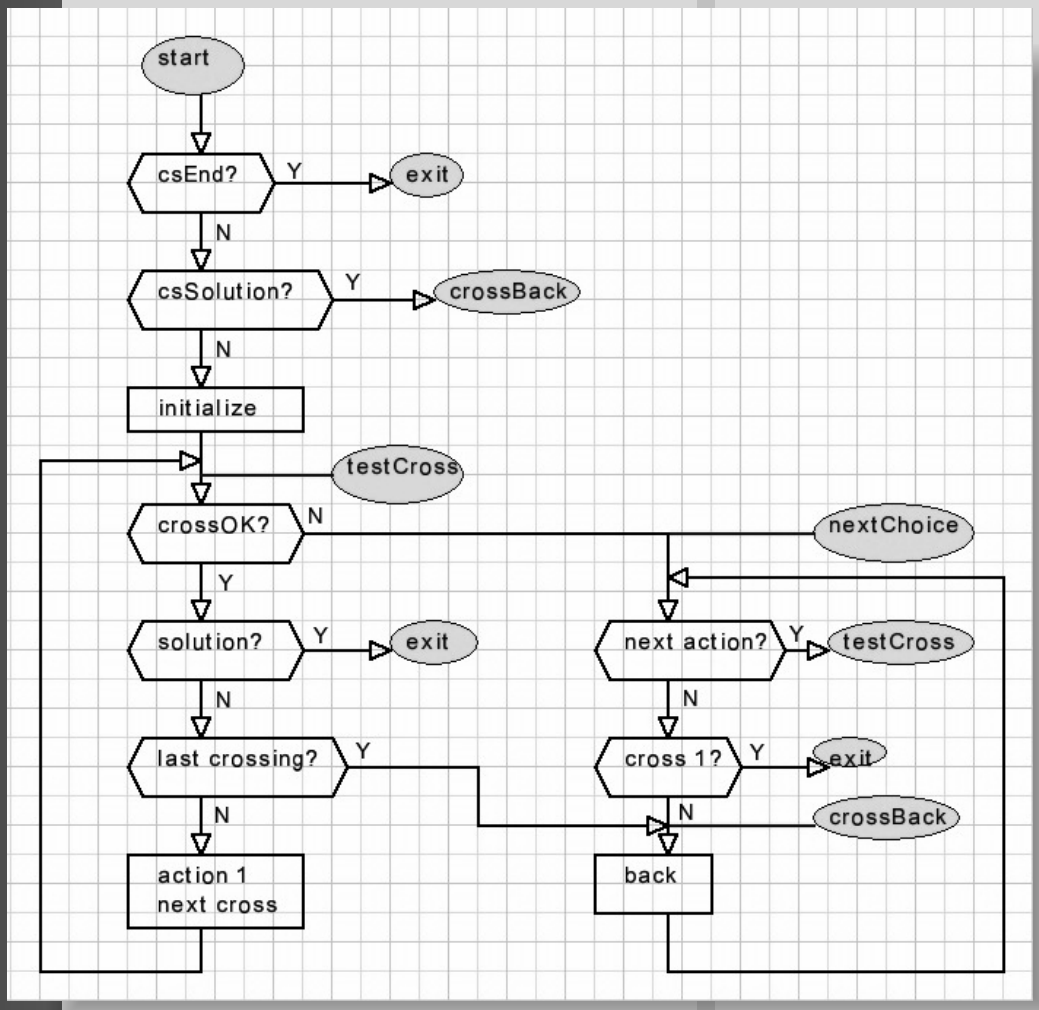
Now, some situations are illegal, such as 0000 0110 meaning that the goat and the cabbage are together and unattended at the right bank.
Procedure generateLegals fills a boolean array called legal. For each situation the array supplies true or false.

During the process no items disappear so the danger is real that items are moved
back and forth endlessly. To prevent this we have to remember previously encountered situations.
Again there is a boolean array history[0..15] with a flag for each situation:
false if not encountered, true if the situation did occur during a previous move.

The solution of this puzzle simply is the list of actions (1..4) in array crossList[ ].
CrossList actually serves as a counter.
Note:
do not be confused by the terms action and actioncode.
action counts 1,2,3,4,1,2,3,4,... and actioncode[action] supplies the value to be xored with the situation.
Initially the contents of crossList is (0 0 0 0 0 0 0 0 0 0 0...) indicating that no action took place yet.
crossNr = 0. situation = 0.
If the farmer and the goat move right :
crossNr = 1 and crossList[ ] = (1 0 0 0 0 0 ....) .
Now situation = 0000 0000 xor 0000 1010 = 0000 1010.
When the farmer returns:
crossNr = 2; situation = 0000 1010 xor 0000 1000 = 0000 0010
crossList[ ] = (1,4,0,0...).
The puzzle is solved when situation 15 (binary 1111) is met : all items on the right bank.

Each action must be tested before because
- only items can be moved that are positioned at the proper bank
- illegal situations must be avoided
- previously encountered situations may not re occur

Now the core of this project: the search for a solution.
This is done by
function FindSolution(cs : TcrossStatus) : TCrossStatus;
where the crossStatus is csEND (no solution) or csSolution.
Below is the flowchart.
Looking at the source code may be shocking for programming purists:
it uses labels and goto statements.
The reason is crossing loops. Using repeat or while controlled loops would need extra boolean variables making the code harder to understand.
If you can make readable code without the goto statements please let me know.



Please refer to the flowchart above for some details.

The function is called with csStart, csSolution or csEnd.
In case of csEnd the function exits immediately: there are no more solutions.
In case of csSolution the search goes for more solutions.
A jump is made to "crossBack".
With csStart, the search for a fresh solution begins.
Initialization:

```
 crossNr := 1;
 ac := 1;
```

If this pass is OK, the cross is registered:

```
 crossList[crossNr] := ac;
 situation := situation xor actionCode[ac];
 history[situation] := true;
```

Test for a solution

```
  if situation = maxSit then
  begin
   result := csSolution;
   exit;
  end;
```

Note: maxsit = 15, binary 1111, the maximum situation.
If no solution a test is made for the maximal pass reached:

```
  if crossNr = maxCross then goto crossBack;
```

If more passes are allowed

```
  inc(crossNr);
  ac := 1;
  goto testCross;
```

The next pass is selected and the ac counter is reset to 1.

If an action is tested false for some reason, the next action is tried: (label nextchoice)

```
 nextChoice:
  if ac < maxActionCode then
  begin
   inc(ac);
   goto  testCross;
  end;
```

If all actions were tried, the pass has to be taken back.
However, if we are forced to take back pass 1, this means there is no solution.

```
  if crossNr = 1 then
  begin
   result := csEnd;
   exit;
  end;
```

If not pass 1, the last pass must be removed:

```
 history[situation] := false;
 situation := situation xor
actionCode[crossList[crossNr]];
 crossList[crossNr] := 0;
 dec(crossNr);
```

This takes us back at the previous pass.

Now we enter code at the label crossBack
This pass was not successful so we have to take it back and try the next action:

```
 crossBack:
  history[situation] := false;
  ac := crossList[crossNr];
  situation := situation xor actionCode[ac];
  crossList[crossNr] := 0;
  goto nextChoice;
```

Next some notes at the function crossOK which tests trial actions.
First test is whether the items are at the proper side of the river:

```
 code := actioncode[ac];
 pos := situation and code;                    //test
                                  //items on proper bank
 if (nr and 1) = 1 then result :=  (pos = 0)   //nr is
                                //pass number (odd,even)
  else result := (pos = code);
```

For a pass from left to right bank, all items must be 0.
For a return path they must be 1.
For a pass from left to right bank, all items must be 0.
For a return path they must be 1.

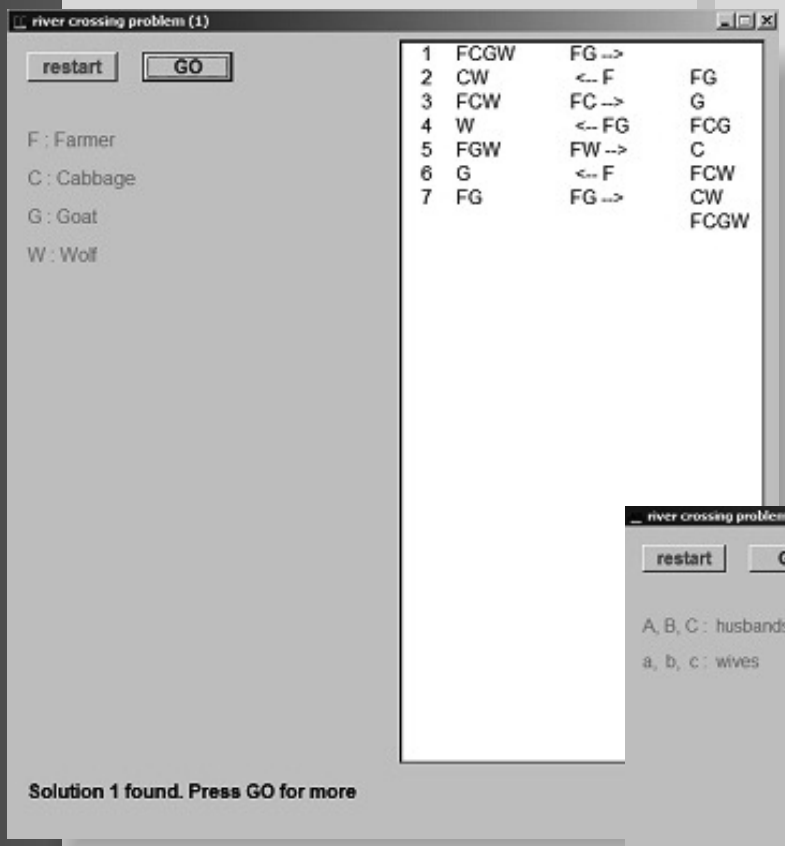
Next test is for an illegal situation:

```
 newsituation := situation xor code;
 if result then
  result := legal[newsituation];
```

Last test is a check if this new situation occurred before:

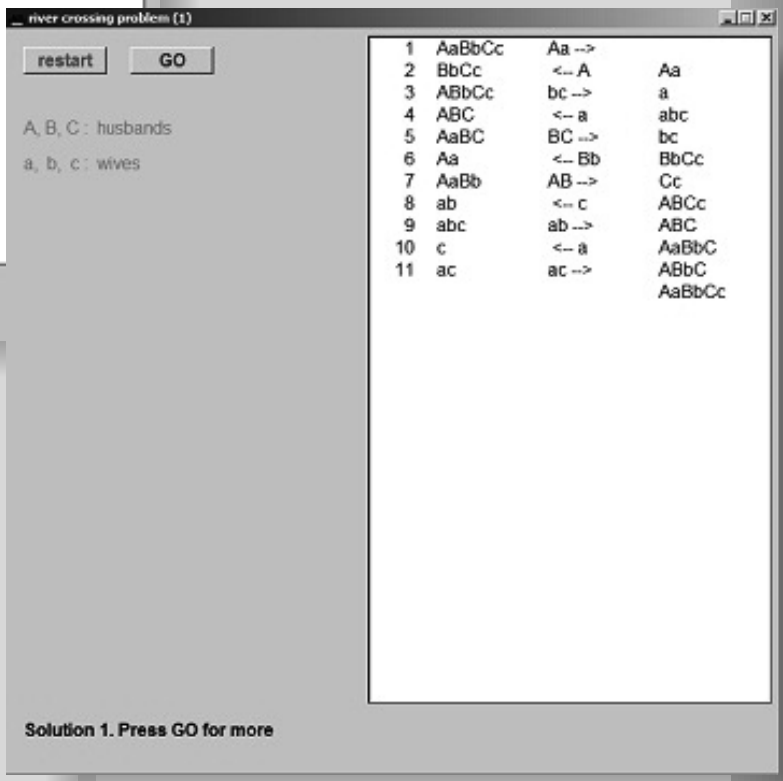
```
 if result then result := (history[newsituation] =
false);//avoid repetition
```

Below a reduced picture of the program + solution:



Procedure generateLegals sets the legal array entry to true or false.
See the source code for the details.
procedure generateActionCodes fills the actionCode array and updates maxAction, the number of actions.
The boat holds one or two people and combinations of husband and wife of different couples is not allowed.
First all 2 people actions are added to the actionList, then all single person actions.
Please refer to the source code for details.

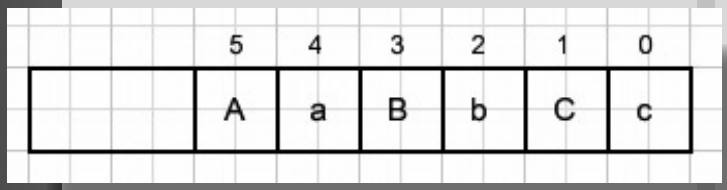
Below is a reduced picture of the program + solution:



So far for rivercross1.

Rivercross2
The differences with rivercross1 are:
1. there are more items
2. the illegal situations are diferent
3. the actions are different
items:

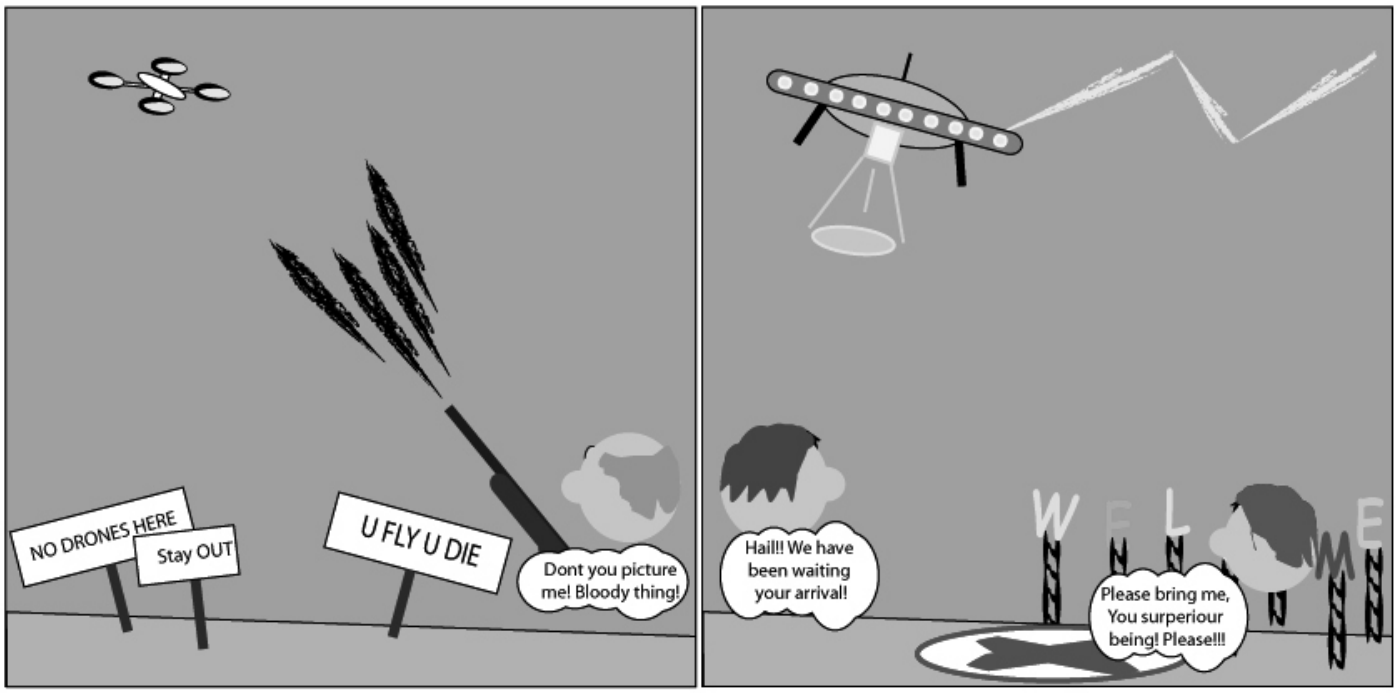
The couples are named Aa
(A male, a female) Bb and Cc.
So there are 64 situations: 0 to 63,
binary 000000 to 111111



Different kind of logic

Di Kim Madsen

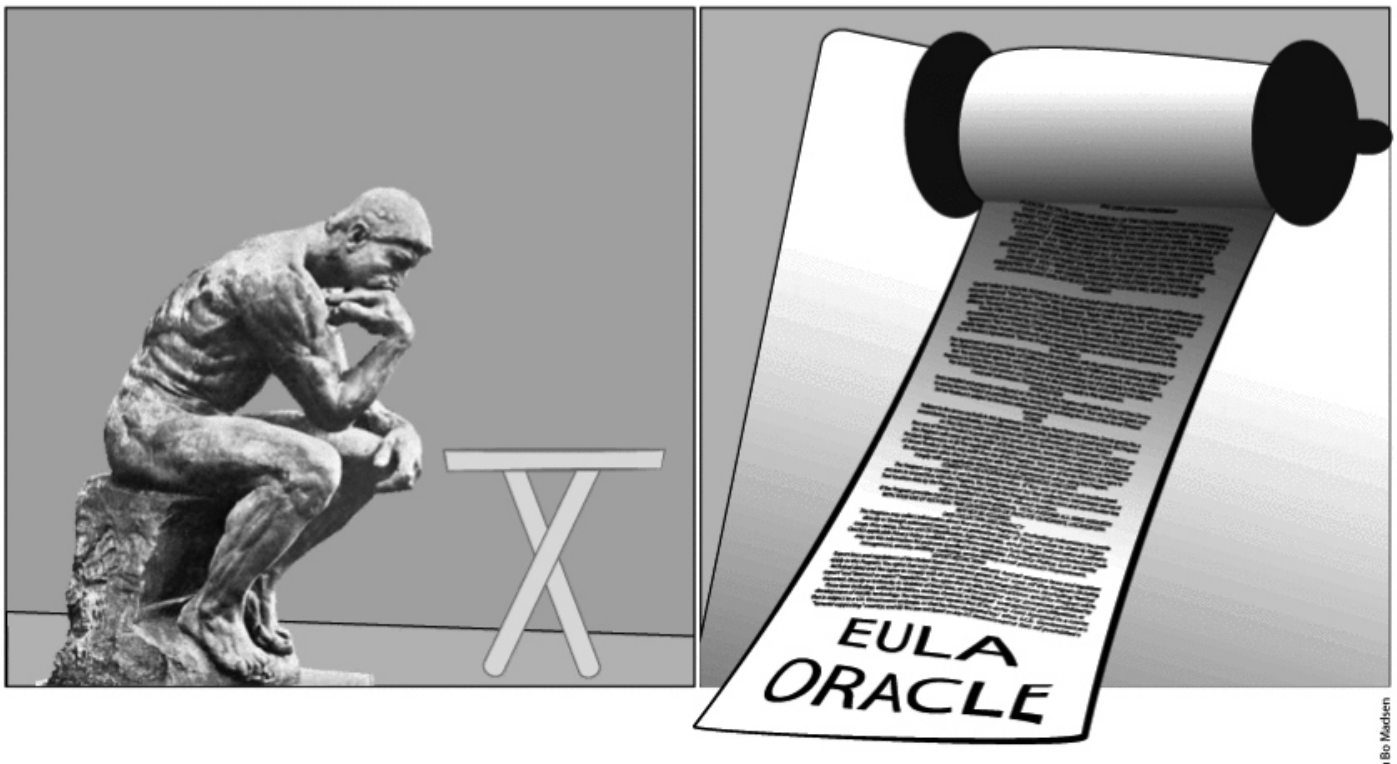
Nerd reflections #11



Kim Bo Madsen

What was Socrates pondering about?

Nerd reflections #12



Kim Bo Madsen

REST EASY WITH KBMMW PART 6
DATABASE 2 BY KIM MADSEN

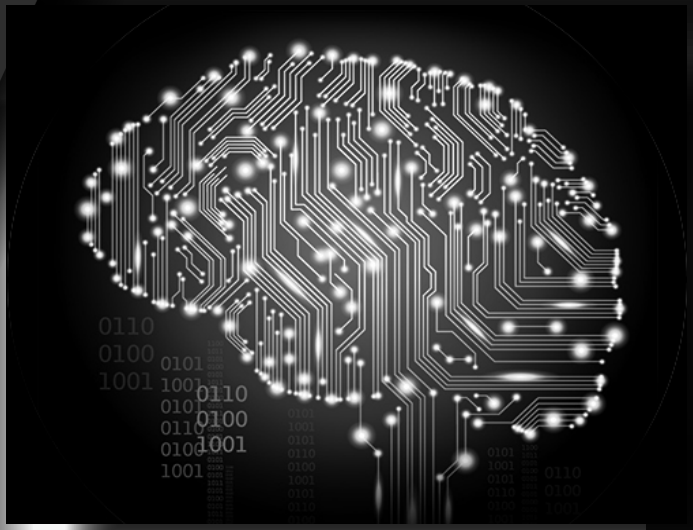
starter expert DX Delphi

Now our fine REST server has been running for some time, and we start to understand we need to expand it with some more data.

Adding an additional table for new info is easy, as it´s done the same way as shown in REST easy with kbmMW 2 – Database

However what if we need to add additional fields to the TContact class? What do we do with the data storage?

Until upcoming release of kbmMW, we would have had to make our own table update mechanism, which often is easy to do, as long as you add new fields, and those fields should not be part of primary keys and such.



6

But from next release of kbmMW, we also release a new beta feature in the kbmMW ORM.

The ability to determine wtherthe data storage is compatible with the class, and the ability to automatically update the data storage to match the new look of the class.

It sounds so deceptively simple to do so,
but when we also want it to work across different databases, handling indexes and more, it suddenly starts to require quite detailed understanding of databases and their metadata.

So as a spin off of this new feature, **kbmMW** also comes with vastly improved **database metadata handling**, even better understanding of SQL query variants, more features in our in house SQL parser and much more.

Ok.. enough of the sales talk….
How do you do it then?
The original Tcontact class was defined like here :

```
unit Unit9;

interface

uses
 DB, System.Generics.Collections,
 kbmMWRTTI, kbmMWORM, kbmMWNullable;

type

[kbmMW_Table('name:contact')]
TContact = class
private
 FID:kbmMWNullable;
 FName:kbmMWNullable;
 FAddress:kbmMWNullable;
 FZipCode:kbmMWNullable;
 FCity:kbmMWNullable;
 FComments:kbmMWNullable;
public
 [kbmMW_Field('primary:true, generator:shortGuid',ftString,40)]
 property ID:kbmMWNullable read FID write FID;

 [kbmMW_Field('name:name',ftString,50)]
 property Name:kbmMWNullable read FName write FName;

 [kbmMW_Field('name:address',ftString,80)]
 property Address:kbmMWNullable read FAddress write FAddress;

 [kbmMW_Field('name:zipCode',ftInteger)]
 property ZipCode:kbmMWNullable read FZipCode write FZipCode;

 [kbmMW_Field('name:city',ftString,50)]
 property City:kbmMWNullable read FCity write FCity;

 [kbmMW_Field('name:comments',ftMemo)]
 property Comments:kbmMWNullable read FComments write FComments;
end;

implementation

initialization
 kbmMWRegisterKnownClasses([TContact,TObjectList]);
end.
```

TkbmMWORMCompatibleTableFlags is a set of flags including:

- **mwoctfBasic** – Basic strictness. Translates to **mwfdctDataType, mwfdctPrecision, mwfdctSize**
- **mwoctfStrict** – Strongest strictness. Translates to **mwfdctPrimary, mwfdctUnique, mwfdctRequired, mwfdctDataType, mwfdctStrictPrecision, mwfdctStrictSize**
- **mwoctfConstraints** – Include constraints validation. Translates to **mwfdctPrimary, mwfdctUnique, mwfdctRequired**
- **mwoctfType** – Include generic data type validation. Translates to **mwfdctSize, mwfdctPrecision, mwfdctDataType.**

The translated comparing flags can't be provided directly, but is used internally, and only shown for completeness.

- **mwfdctPrimary**
  – Primary key definition must match.
- **mwfdctUnique**
  – Unique field constraint must match.
- **mwfdctRequired**
  – Required field constraint (not null) must match.
- **mwfdctDataType**
  – Exact data type must match.
- **mwfdctStrictPrecision**
  – Field precision must match exactly. If not specified the data storage may have a larger field precision than required.
- **mwfdctStrictSize**
  – Field size must match exactly. If not specified the data storage may have a larger field size than required.

Walking thru the issues table can be fun, but even more fun would be not to have to do so.

```
var
 sl:TStringList;
begin
 sl:=FORM.GetUpgradeTableStatements(TTable2);
 try
  // It will make a strict comparison (arguments can be added to choose
non strict comparison),
  // and generate a list of statements in generic kbmMemTable SQL
format
  // that can be used to transform the data storage to be compatible with
  // the class.
  // It could be ALTER TABLE xxx DROP COLUMN yyy
 finally
  sl.Free;
 end;
```

Now the observant reader may say: "That´s all fine, but I for a fact know that SQLite does not support **ALTER TABLE DROP COLUMN** statements! So it won´t work!"

You are right…. about the SQLite limitation. However remember that kbmMW will translate the statements into something acceptable by the target database type, so SQLite will in fact suddenly be able to have a column dropped from a table containing data. kbmMW will do its best to make it happen.

If you would like to see the rewritten SQL. In other words generic kbmMemTable SQL converted to specific target database syntax, then do like this:

```
var
 sl:TStringList;
begin
 sl:=FORM.GetUpgradeTableStatements(TTable2,false);
 try
  // Now the list of statements will have been converted to
  // the specific target database.
 finally
 sl.Free;
 end;
```

And after all this gibberish then how to make the data store compatible with the new class?

```
FORM.UpgradeTable(TTable2);
```

After running this, the table "contact" in the database will have been made compatible with the class, with all remaining data retained.

If you have huge tables with billions of rows, then it might be better to get inspiration from the output from GetUpgradeTableStatements, and apply the changes under human supervision.

Although kbmMW attempts to do things in a safe way, I also recommend backing up the data storage before attempting an automatic upgrade.

Currently kbmMW contains SQL rewriters that targets **SQLite, MySQL/MariaDB, PostgreSQL, MSSQL 2008/2012+, Oracle 9+, Interbase/Firebird** and generic **SQL92** and **SQL2003** compatible databases.

When the beta of this upgrade mechanism is released we urge people to test it upgrading capabilities carefully before deploying to production.

best regards

Kim / C4D

Let´s add a Gender field, change the Name field to be unique (*just for fun*), and change the zip code field to be a string type matching the property type (*previously we, perhaps incorrectly, defined it as an integer data storage field, where storing it as a string might have been better*).

```
unit Unit9;

interface

uses
 DB,
 System.Generics.Collections,
 kbmMWRTTI,
 kbmMWORM,
 kbmMWNullable;

type

[kbmMW_Table('name:contact')]
TContact = class
private
 FID:kbmMWNullable;
 FName:kbmMWNullable;
 FAddress:kbmMWNullable;
 FZipCode:kbmMWNullable;
 FCity:kbmMWNullable;
 FGender:kbmMWNullable;
 FComments:kbmMWNullable;
public
 [kbmMW_Field('primary:true, generator:shortGuid',ftString,40)]
 property ID:kbmMWNullable read FID write FID;

 [kbmMW_Field('name:name, unique:true',ftString,50)]
 property Name:kbmMWNullable read FName write FName;

 [kbmMW_Field('name:address',ftString,80)]
 property Address:kbmMWNullable read FAddress write FAddress;

 [kbmMW_Field('name:zipCode',ftString,20)]
 property ZipCode:kbmMWNullable read FZipCode write FZipCode;

 [kbmMW_Field('name:city',ftString,50)]
 property City:kbmMWNullable read FCity write FCity;

 [kbmMW_Field('name:gender',ftString,1)]
 property Gender:kbmMWNullable read FGender write FGender;

 [kbmMW_Field('name:comments',ftMemo)]
 property Comments:kbmMWNullable read FComments write FComments;
end;

implementation

initialization
 kbmMWRegisterKnownClasses([TContact,TObjectList]);
end.
```

Usually these changes in the class would render that class incompatible with the data storage. In fact we can now ask the **ORM** if the data storage is compatible with the class we have.

```
if not FORM.CompatibleTable(TContact) then
 raise Exception.Create('Tcontact is not compatible with the datastore');
```

When we call CompatibleTable this way, it compares using the strictest `comparison method`, which means that storage fields and index definitions must be not only compatible, but identical. By adding an `TkbmMWONObject` instance to the call, we can be told what problems there are:

```
var
 issues:TkbmMWONObject;
begin
 issues:=TkbmMWONObject.Create;
 try
  if not FORM.CompatibleTable(TContact,issues) then
  begin
   // Decipher issues object.
   // There may be 3 properties in the object, named add, modify, delete
   // and each of those will be an array of objects with properties for name,
                    unique, primary, required, size, precision and dataType.
  end;
 finally
  issues.Free;
 end;
end;
```

This way you get detailed information about the changes needed to make the data storage compatible with your class.
It is possible to tune exactly what to compare and how, and thus limit the strictness of the comparison mechanism. This is done by adding one additional argument to `CompatibleTable`, namely the `ACompatibleFlags`: **`TkbmMWORMCompatibleTableFlags`**.

REST EASY WITH KBMMW PART 7
CONFIGURATION BY KIM MADSEN

starter expert DX Delphi

The upcoming release of kbmMW will also contain a brand new configuration framework.

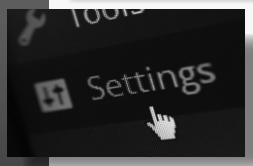
What is a configuration framework?

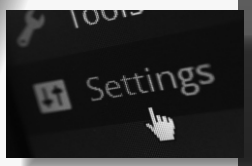
You have probably already used one many times, in the form of reading or writing data to/from INI files, registry settings, your own XML files etc.

Basically most applications need to obtain some user defined configuration from somewhere.

In kbmMW we have until now been doing it the same way as everybody else, but it is so boring and tedious to handle configurations the old fashioned way.

Why do we have to write all that boiler plate code to read some data from a configuration, or to store something back into the configuration, and doing that we usually forget doing backups and such, risking to damage a working configuration, if we experience an unexpected power failure at the wrong time. But that is all past now





CONFIGURATION

kbmMW's new `TkbmMWConfiguration` class and its accompanying storage classes descending from **`TkbmMWCustomConfigurationStorage`** will automate all the tedious work for you.

So how to use it?

The simplest way possible to read a configuration:

```
unit Unit1;

interface

uses
 ...,
 kbmMWConfiguration;

procedure TYourClass.DoSomething;
begin
 SomePath:=
   Config.AsString['myconfig.somepath'];
   Config.AsString['myconfig.somepath']:=
                    'SomeNewPath';
end;
```

As you can see, you will have a thread safe `Config` instance readily available for you, and we are using it to read some string out, and store a string back.

So where is this configuration then persisted? It will default create and access an **`XML`** file named **`yourapplicationname_config.xml`** placed in the startup directory of your application. Its contents, given the above example, would look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
 <myconfig>
  <somepath>SomeNewPath</somepath>
 <myconfig>
</config>
```

As you can see, it will use the dot notation of your keys to allow you to group your configuration settings anyway you want.

Obviously there are several additional methods to `AsString` which can be used for fetching or storing other types of data.

Currently **kbmMW** provides:

- **AsInt32**
- **AsInt64**
- **AsDouble**
- **AsBoolean**
- **AsDateTime**
- **AsBinary**
- **AsString**

And some functions to easily return a default value, in case nothing is defined in the

They all follow the same syntax like this one:

```
function AsDefString(const APath:string;
          const ADefault:string;
          const ASetIfDefault:boolean = false):string;
```

Notice the `ASetIfDefault` value. It is default false, but if you set it to true, the configuration storage will be updated with the default value, if no value is found for the key given by `Apath`.

The other functions are

- **AsDefInt32**
- **AsDefInt64**
- **AsDefDouble**
- **AsDefBoolean**
- **AsDefDateTime**
- **AsDefBinary**

You can test for wether a value exists in the configuration file by using the contains property:

```
if Config.Contains['myconfig.somepath'] then...
```

The configuration file is automatically opened and loaded the first time you use any of the above properties/functions.

NOTICE the distinction between opened and loaded! If you remember the good old **INI file**, then you open it, and every time you read from it, you will read from the actual file, and thus immediately see changes without the need to reopen it. The same when using the Windows registry.

That may be a nice thing, or it may be wasted CPU cycles depending on the scenario. If you read a value lots of times, you would usually put the configuration into a variable that you manage, not to waste too much energy reading it from the **INI/Registry**.

CONFIGURATION STORAGES

How does kbmMW do it? It depends on which of the configuration storage's that is being used. kbmMW currently supports the following storage's:

- **TkbmMWIniConfigurationStorage**
- **TkbmMWRegistryConfigurationStorage**
- **TkbmMWJSONConfigurationStorage**
- **TkbmMWXMLConfigurationStorage**
- **TkbmMWYAMLConfigurationStorage**

The Ini and Registry configuration storage operates live on the INI file and the Registry entries, as you would normally expect.

The remaining configuration storage's operates on an internal representation of the configuration, which will be streamed back into the file at relevant times.

What is a relevant time?

- Upon application shutdown if anything has been changed
- Upon calling Save method on the storage if anything has been changed
- On every change, if AutoSave property is true

If you use the **XML, YAML** or **JSON** storage, you will even have the advantage of **kbmMW** automatically backing up your old configuration before saving the new. You can control how many backup files you want to leave via the BackupMaxCount (*default 5*) property, and where it should be backed up to via the BackupPath (*default same path as original file*) property and finally what extension the backup file should have via the BackupExt (*default 'bak'*) property.

The **XML, YAML** and **JSON** storage methods all stores the data in a tree structure following the dotted names of the given path.
The Ini storage method use the first segment of the path as the section name, and the remaining part of the path (*including dots*) as the key name. Eg.

```
Config.AsInt32['section1.b.def']:=10
```

Will result in an **ini file** like this:

```
[section1]
b.def=10
```

The Registry storage method use the last segment of the path as the name value and the remaining as the registry key tree path. In addition one specify the RootKey (*defining which registry hive to operate*) and the starting place in the tree within the hive when creating an instance of the storage. If you want to change the storage method of the configuration, you do like this before using any of the access methods:

```
Config.Storage:=T
kbmMWRegistryStorage.Create( HKEY_LOCAL_MACHINE,
        'Software/MyCompany');
```

With the above registry storage, setting Config.AsInt32['a.b.c.def']:=10 would result in opening the registry key Software/MyCompany/a/b/c in the registry hive HKEY_LOCAL_MACHINE, and setting the value def to the integer value 10. kbmMW automatically opens and closes registry keys as needed.

SPLIT CONFIGURATION

You have seen how easy it is to access the standard Config object. But what if you explicitly want two (*or more*) different configurations concurrently?
Solution: Instantiate your own **TkbmMWConfiguration** instances. Eg.

```
var
 myConfig1,
 myConfig2:IkbmMWConfiguration;
begin
 myConfig1:=TkbmMWConfiguration.Create;
 myConfig1.Storage:=
    TkbmMWXMLConfigurationStorage.Create(
       'myconfiguration1');
 myConfig2:=TkbmMWConfiguration.Create;
 myConfig2.Storage:=
    TkbmMWXMLConfigurationStorage.Create(
       'myconfiguration2'); ...
end;
```

This way you can also specify your own naming of the configuration, which in the above examples results in **myconfiguration1.xml** and **myconfiguration2.xml** unless you also decide to change the configuration storage method.

myConfig1 and **myConfig2** can then be used the same way as we used the standard Config object.

AUTOMATIC CONFIGURATION

It is now already simple to access the configuration from anywhere in your application, but you can also make it automatic.

Say you have an object which you want to hold certain values based on your configuration. You could yourself instantiate the object and assign the values using the As… properties, but there is also another way.

```
TMyObject =
class(TkbmMWConfigurableObject)
private
 [kbmMW_Config('myconfig.somepath')]
 FSomePath:string;
public
 property SomePath:string read FSomePath;
end;

initialization
 kbmMWRegisterKnownObject(TMyObject);
```

By descending your object from `TkbmMWConfigurableObject`, and using the `kbmMW_Config` attribute on either properties or fields, your configuration will automatically be read and optionally written when you instantiate or destroy your object.

Upon instantiating `TMyObject`, then the property SomePath would have the value 'SomeNewPath', provided this blog previous examples were followed.

Default the fields/properties with the attribute on are only read from the configuration storage. If you want the configuration to be automatically updated with new values from your object, then you need to provide an additional argument to the attribute:

```
[kbmMW_Config('myconfig.somepath',mwcdReadWrite)] FSomePath:string
```

You can also define if date/time values are read and stored as UTC *(default)* or local time values, by setting yet another argument.

```
[kbmMW_Config('myconfig.sometime',mwcdRead,false)] FSomeTime:TkbmMWDateTime
```

kbmMW also supports `FSomeTime` being a regular `TDateTime`.

You can force the reload of the configured properties by calling the `ReadConfig` method of your configuration object, and you can force saving the properties that supports being saved by calling `WriteConfig`.

You can switch which configuration to load the configuration from by setting the Configuration property of your object to point to a relevant **kbmMW configuration instance** and explicitly call ReadConfig.

What if you already have an object hierarchy and it is not possible for you to inherit from `TkbmMWConfigurableObject?` Then you can still use kbmMW's smart configuration by manually requesting the configuration and registering your object as a known object with kbmMW. Eg. now not descending from **`TkbmMWConfigurableObject`**:

```
TMyObject = class
private
 [kbmMW_Config('myconfig.somepath')]
 FSomePath:string;
public
 property SomePath:string read FSomePath;
end;

initialization
 kbmMWRegisterKnownObject(TMyObject);
```

To load the configuration do:

```
myobject:=TMyObject.Create;
Config.ReadConfig(myobject);
```

and saving it

```
Config.WriteConfig(myobject);
```

If you want to make it automatic for your own object, override the methods `AfterConstruction` and BeforeDestruction of your object class.

```
…
public
 procedure AfterConstruction; override;
 procedure BeforeDestruction; override;
…
```

SMART SERVICES

Finally, if you use kbmMW smart services, you can also automatically use **kbmMW** configuration data as arguments for the service calls. Eg:

```
[kbmMW_Method('EchoReversedConfigString')]
 [kbmMW_Rest('method:get, path:
              "echoreversedconfigstring"')]
// This method obtains its value from the configuration.
 function ReverseStringFromConfig([kbmMW_Config(
      'a.b.c.value')] const AString:string):string;
```

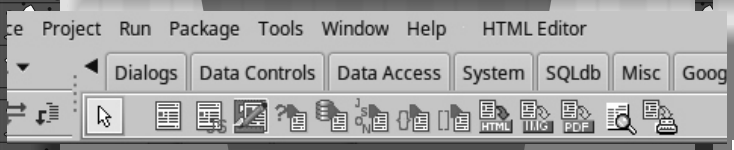
Upon calling the `echoreversedconfigstring` method its **`AString`** argument will always contain a string value read from the configuration.

This concludes this article about the new configuration features in the upcoming kbmMW release.

starter expert

ABSTRACT
In this second article we delve deeper in the possibilities of FPReport: we show how to save and load a design, make groups in our report, and how to display totals in footers or headers of these groups. We end with the visual report designer.

INTRODUCTION
In a previous article, the design of the FPReport reporting engine was described. A simple report was created, and the use of an exporter to create for example a PDF file of a text file was demonstrated. In this article, we'll demonstrate how to save and load a report design from file, and how this can be combined with the visual designer that comes with FPReport.
The previous article showed that a simple list - even a multi-column one - is easy to do. Grouping is not more difficult, and in this article, we'll show how this can be accomplished. We'll also show how to work with variables to define and print totals.
Although building of reports in code is enlightening and provides understanding of how the engine works, it can be quite tedious. So, a visual report designer is an absolute must, and fpreport comes with one. The highlights of the designer will also be shown - it is in fact pretty standard for a designer.



SAVING AND LOADING A REPORT FROM STREAM
Creating a report design in code is cumbersome. A report design can be created visually, and the design can be saved to file. FPReport relies on a helper class to save/load a report from file: A descendent of TFPReportStreamer. This is done so various formats can be supported. The default format is JSON, but any format (XML, YAML) could be implemented.
The following will save the report to file, once it was designed:

```
procedure TPrintApplication.SaveReportDesign;
Var  J : TFPReportJSONStreamer;
 F : TFileStream;  S : TJSONStringType;
begin
 F:=Nil;
 J:=TFPReportJSONStreamer.Create(Self);
 try
  FReport.WriteElement(J);
  F:= TFileStream.Create('txt2pdf.fpr',fmCreate);
  S:=J.JSON.FormatJSON();
  F.WriteBuffer(S[1],Length(S));
 finally
  F.Free;
  J.Free;
 end;
end;
```

This is, in fact, quite simple; a streamer instance is created, the report is saved to the streamer, and the resulting JSON is written to a file. Nothing could be simpler.
The following code does the reverse operation: it loads the report from a file:

```
procedure TPrintApplication.LoadReportDesign;
Var  J : TFPReportJSONStreamer;
 F : TFileStream;  O : TJSONObject;
begin
 J:=Nil;
 F:=TFileStream.Create('txt2pdf.fpr',fmOpenRead);
 try
  O:=GetJSON(F) as TJSONObject;
  J:=TFPReportJSONStreamer.Create(Self);
  J.JSON:=O;
  J.OwnsJSON:=True;
  FReport.ReadElement(J);
 finally
  F.Free;
  J.Free;
 end;
end;
```

Again, nothing spectacular. Note the OwnsJSON:=True, this tells the streamer it should free the JSON object when it is freed itself. Note that for this to work, some small changes are needed to the sample program of the previous article. The first change is that the data loop object must have a name, and must be registered with the report. This is done in the constructor of our application object:

```
constructor TPrintApplication.Create(
Aowner: TComponent);

begin
 Inherited;
 FReport:=TFPReport.Create(Self);
 FLines:=TStringList.Create;
 FData:=TFPReportUserData.Create(Self);
 FData.Name:='Data';
 FReport.ReportData.AddReportData(FData);
```

The reason for this is that when writing the report design, the report elements will write the name of the data loop to the stream. When reading the report design from stream, they will use this name to look up the data loop in the list of registered report data loops: for this reason, the data loop needs a name and must be registered in the reportdata collection.
Since the name of the data loop is set, the variable name in the memo that prints the actual line, must be changed:

```
M:=TFPReportMemo.Create(DB);
 M.Text:='[Data.Line]';
```

The name consists of two parts, separated by a dot: the data loop name, and the variable name. A report can have multiple data loops, so the name of the data loop becomes important.
A version of the example program that uses a design in a separately stored file is also available. The `fpjsonreport` unit contains a `TFPJSONReport` descendent which does all the above:

```
TFPJSONReport = class(TFPReport)
 procedure LoadFromStream(const aStream: TStream);
 procedure SaveToStream(const aStream: TStream);
 Procedure LoadFromJSON(aJSON : TJSONObject); virtual;
 Procedure SavetoJSON(aJSON : TJSONObject); virtual;
 Procedure LoadFromFile(const aFileName : String);
 Procedure SaveToFile(const aFileName : String);
end;
```

Why this elaborate design, why not simply have two methods in `TFPReport :LoadFromFile` and `SaveToFile` ? Several reasons, in fact:

- **One** is an architectural pattern, separation of concerns: the report class should not be concerned with the file format. This allows to save to any desired format. **JSON** is the default choice but an *(untested)* XML streamer also exists. Each can use the format he/she chooses.
- **The second** is that the report stream only contains the design of the report. A standalone reporting tool that fetches data needs to add information to the stream about the data for the data loop: server connection data, the SQL to execute, parameters to ask etc.
  The standalone **FPReport designer** tool uses this mechanism. By separating the streaming from the core reporting tool, it becomes possible to hook into the streaming mechanism and save whatever additional data is needed.

GROUPING

Many reports will do some kind of grouping. For sales results or forecasts, the monthly, quarterly or even yearly totals are a must. For an itemized invoice, the items may be grouped in categories, or according to VAT tariff. A list of countries may be broken down in continents or even parts of a continent, or simply by grouping them according to the first letter of their name.
To create a group in a report requires 3 steps:

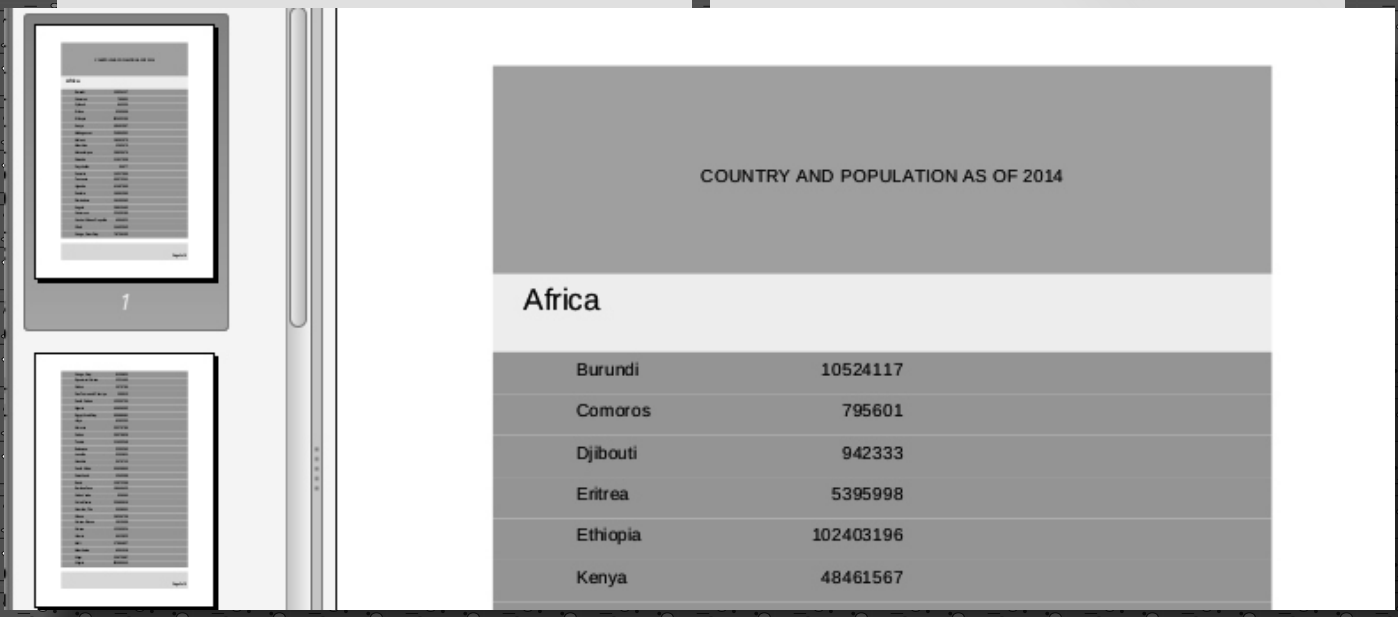
1. Determine a grouping condition.
This is an expression that, when it changes, signals the start of a new group. In the example of the list of countries, this expression can be simply the name a field in the data that contains the continent name, or an expression: `Copy(Country,1,1)`. This returns the first letter of the name of the country.

2. Sort the data so the items in the data list are grouped together.
fpReport will not do this for you, you must take care of this yourself. In the example of the list of countries, the list of countries must be sorted first on the continent, or alphabetically - depending on what grouping mechanism you want to use.

3. A group header (*a band of class TFPReportGroupHeaderBand*) must be inserted in the report, and it's **GroupCondition** must be set to the expression that determines the group. This must be done, even if you only want to display group totals at the end of a group. (*you can set the header's visible property to False, or set the height to zero*).
So, in the case of a list of countries, the following code will add a group header, which displays the continent of the countries that follow:

```
GroupHeader := TFPReportGroupHeaderBand.Create(p);
 GroupHeader.Layout.Height   := 20;
 GroupHeader.GroupCondition :=
                        'Continent)';
 Memo :=  TFPReportMemo.Create(GroupHeader);
 Memo.Layout.Top      := 5;
 Memo.Layout.Width    := 10;
 Memo.Layout.Height   := 8;
 Memo.UseParentFont := False;
 Memo.Text            := '[Continent]';
```

Figure 1: Report with grouping



DISPLAYING TOTALS AND OTHER AGGREGATES

Grouping the items makes a list not only more easily to read, it can also be used to display additional information: a group total, for example, the total population of the countries per continent. There are two ways to display such a total: A simple one, which is only usable in the group footer, the other - involving an extra variable - can be used both in the group header and the group footer. We'll start with the simple one, which can only be used on a group footer. In the below code we'll display the total population of the continent at the end of the continent. This is done in a group footer (*a band of type TFPReportGroupFooterBand*).

```
// Create the group header
GroupHeader := TFPReportGroupHeaderBand.Create(p);
GroupHeader.Layout.Height := 20;
GroupHeader.GroupCondition := 'Continent';
// Create the group footer
GroupFooter := TFPReportGroupFooterBand.Create(p);
GroupFooter.Layout.Height := 20;
// Attach to the correct group
GroupFooter.GroupHeader:=GroupHeader;
Memo := TFPReportMemo.Create(GroupHeader);
Memo.Layout.Left := 15;
Memo.Layout.Top := 5;
Memo.Layout.Width := 10;
Memo.Layout.Height := 8;
Memo.UseParentFont := False;
// Display the total.
Memo.Text := 'Total for [data.continent]: '+
     '[FormatFloat(''#,###0.00'', '+
     ' sum(data.population/1000000))] million.';
```

The two things to take note of in the above code is first of all the fact that the group footer must be attached to the group header with its `GroupHeader` property. This is necessary to establish the logical structure of the report: if multiple groups and group footers and group headers are present, the reporting engine has no way of knowing what footer belongs to what footer - bands have no 'position' in the report which could aid in determining the logical structure.

The second thing to note is the use of the Sum aggregate function in an expression.

The reporting engine expression parser knows that this function is an aggregate expression: whenever the loop changes record, it will update all aggregate functions.

There are multiple aggregate functions:

- **sum** calculates a simple sum of its argument.
- **avg** calculates a simple average of its argument.
- **max** Displays the maximum value of its argument.
- **min** Displays the minimum value of its argument.
- **count** keeps a simple count of the number of times it was updated and returns that count.

Whenever a memo containing an aggregate value was printed, the aggregate value is set. This means that when the memo is displayed in a group footer, the values of the expression will take into account only the records of the group. Should you wish to display e.g. a running total, then the reset of the aggregate values can be disabled through the **`moNoResetAggregateOnPrint`** option of the memo.

Figure 2 shows a screenshot of a report with the total of a continent.

The above is quite simple to do, but also limited in possibilities. It does not allow us to put a total of the group header, since the total isn't known yet when the group is started. Also, when reprinting a group header on a page start, or displaying already a footer at the end of a page, this method will give wrong results. Some memos will need to display expressions that are reset on print combined with expressions that are not reset on print. Therefor an alternative way to calculate and print group totals is introduced, and this is through report variables. The value of Report variables can be used in expressions by simply referencing the name of the value (*you must take care not to use a name of a value in a dataloop, though*).

Report variables exist in 2 kinds:

- **Static variables.**
  Their value is set once, and remains the same until a different value is set.
- **Expression variables.**
  Their value is calculated through an expression and is continuously updated as the reporting engine goes through the data loops.

The expression for this kind of variable can contain an aggregate function, but, more to the point: it can contain a reset expression.

The reset expression determines when an aggregate function in the variable expression is reset. There are several possibilities:

- At the start of a group.
- At the start of a new column.
- At the start of a new page. To use an expression variable to display the total footer in our previous example, the following can be done.

First, the report variable must be defined:

```
Var
 V : TFPReportVariable;
begin
 V:=Report.Variables.Add('PopSum');
 V.Expression:='Sum(Population)'
 V.ResetValueExression:='continent';
end;
```

After this, the expression for a memo displaying the total simply becomes:

```
// Display the total.
Memo.Text := '[PopSum]';
```

Lastly, the **TwoPass** option of the report must be set to **True**: the report will calculate the values for all groups in the first run of the report, and these values are then used in the second run of the report.
The amount of work to display a total in this manner is not so much harder than in the 'simple' way.
To display a total on a footer band at the end of a page, the reset expression simply becomes:

```
V.ResetValueExpression:='PageNo';
```

Needless to say, the same variable cannot be used to display the page total and a group total.
To make life easier, there are even some auxiliary functions to make it easier to create these variables; they compute the reset expression for you.

```
Function AddExprVariable(aName : String; aExpr: String;
          aType: TResultType = rtString;
          aResetType: TFPReportResetType = rtNone;
          aResetGroup: TFPReportCustomGroupHeaderBand = nil)
          : TFPReportVariable;
Function AddExprVariable(aName : String;
          aExpr: String;
          aType: TResultType;
          aResetType: TFPReportResetType;
          aResetValueExpression: String) : TFPReportVariable;
```

The first form of this function can be used like this:

```
rpt.Variables.AddExprVariable('PopSum',
          'sum(StrToFloat(population) / 1000000)',
          rtFloat,
          rtGroup,
          GroupHeader);
```

This function will simply copy the value for the reset expression from the group band groupcondition, plus all parent groups. (*in case of nested groups, all groups must be taken into account*).
With expression variables, new things become possible, such as displaying a running total next to a group total. First, create the variables:

```
Var
 V : TFPReportVariable;
begin
 V:=Report.Variables.Add('PopSumGroup');
 V.Expression:='Sum(Population)'
 V.ResetValueExpression:='Continent';
 V:=Report.Variables.Add('PopSumGroupRunningTotal');
 V.Expression:='Sum(Population)'
 // Reset expression empty !
end;
```

And then show them in a memo

```
// Display the total.
Memo.Text := '[PopSumGroup] (Running
total: [PopSumGroupRunningTotal])';
```

The expression variables are a powerful tool, but care should be taken: do not attempt to use expression variables in an expression for another expression variable; The reporting engine will detect this and give an error.

THE VISUAL DESIGNER

Creating reports in code should not present a problem for even a junior programmer. But it will be tedious and hard work. For an end-user to make a report in this way is of course impossible - disregarding even the fact that the code needs to be compiled.
Luckily, a visual designer with a simple point-and-click interface is also available. It can be used in the lazarus IDE, but can also be integrated in an end-user application to allow the user to design his own reports. There is also a stand-alone version of the designer. For example, when loaded in the designer, the file printing report looks as in figure 3 on the next page. It's clear that for a novice, this is much easier to understand and manipulate than the same code needed to create the report.
The reporting engine is configurable regarding the features it exposes to the user:

- Manage data.
- Manage variables.
- Manage bands.
- Manage pages.
- Load a report.
- Save a report .
- Add pages.
- Start a new report.
- Preview a report.

This means a very restricted version can be given to the user where he is able to set things like font color, position and size, but cannot do anything else: That means he can simply customize a pre-made design.

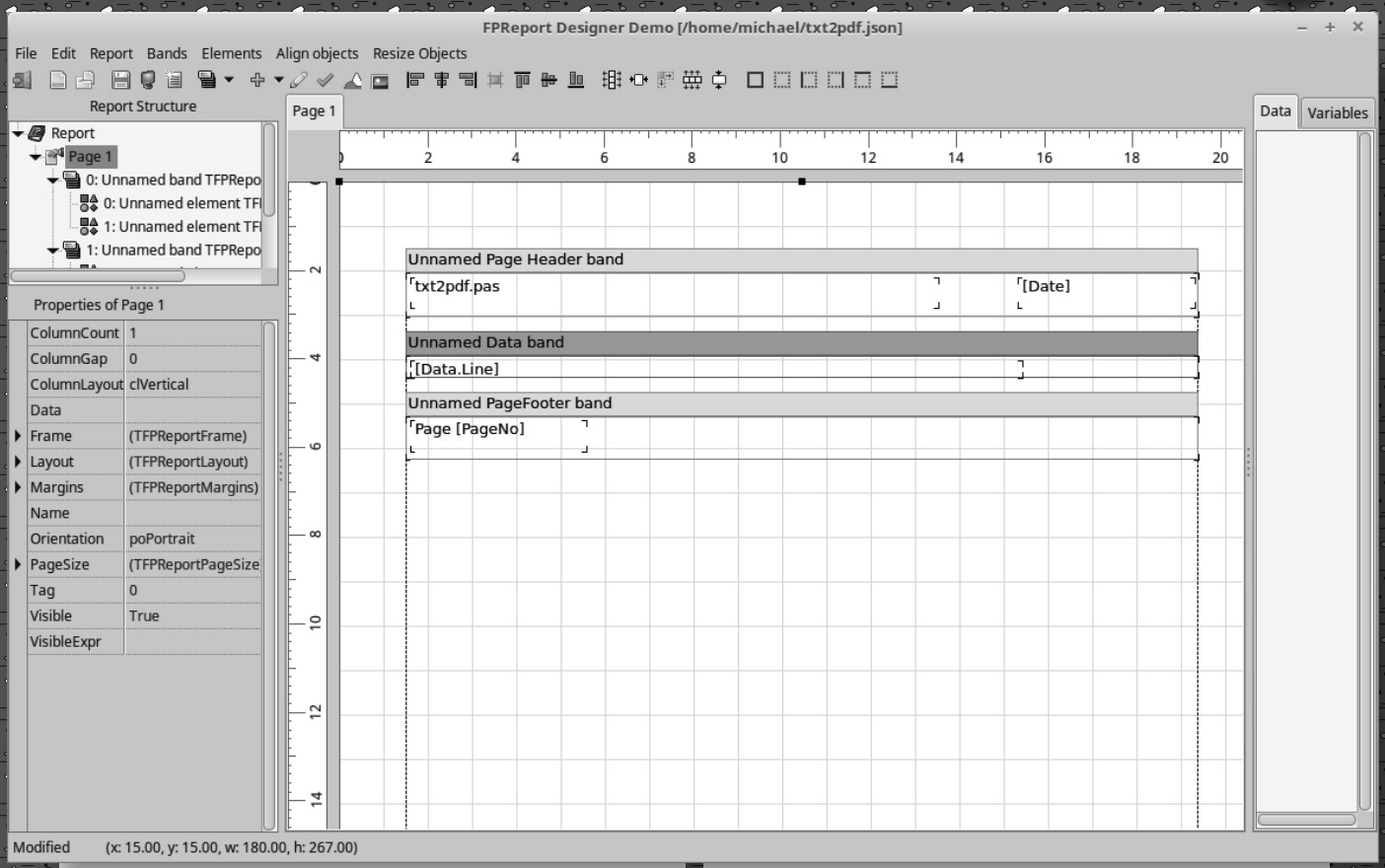


Figure 3: The report design in the designer

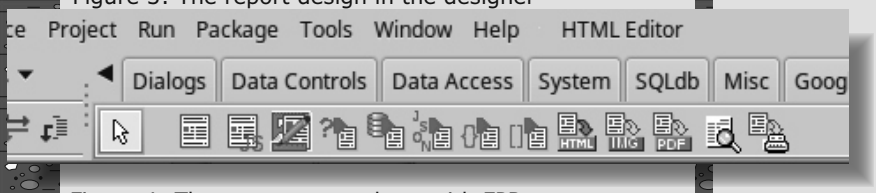


Figure 4: The component palette with FPReport package installed

But at the same time a version can be shipped where the user can do everything: in a large application with extended user management, the amount of options available could be based on the role of the user.
When called from the IDE, the designer does not allow to manage data, since the data must be provided by the programmer using various data loops. The report design can be stored in the lazarus form file if the TFPJSONReportcomponent is used. If it is used to edit a plain TFPReport component, the report has to be stored on disk, and loaded at runtime with some code.
The standalone designer has all options enabled (data choices are stored in the report file it generates).
The main form of the report designer as shown in figure 3 is divided into 5 main areas:

1. The menu and toolbar at the top.
2. The object inspector and report structure on the right.
3. The pages of the report in the middle.
4. The data of the report to the right, it is possible to use drag&drop to drop a variable or field as a new memo to a report.
5. The status bar at the bottom, showing some extra information.

The toolbar can, in addition to the universal new, load and save buttons, be used to quickly set some properties or add some common elements to a report. Less common elements can be added using the 'Element' menu or toolbutton, where entries for all available elements are shown.
Adding a band is done using the band menu, or the band button; A menu item is available for all types of supported bands.
The 'Report' menu harbors the dialogs for adding a page, managing variables or data.

USING THE DESIGNER IN THE IDE

To create a report design in the Lazarus IDE, the **lazidefpreport** package must be installed. The component palette will then be extended with a **FPReport** tab, as shown in figure 4. (left , middle)
The various components are, in the order that they appear on the component palette:

TFPReport
a report component. The design must be loaded from and saved to file.
TFPJSONReport
a report component. The design is stored in the (.lfm) form file.
TFPJSONReport
a report component. The design is stored in the (.lfm) form file.
TFPReportDesigner
The report designer component. This component is only needed if you wish to enable the end user to change the report design.
TFPReportUserData
a report data loop component. The data is obtained through events.
TFPReportDatasetData
a report data loop component. The data is obtained from a dataset.
TFPReportJSONData
a report data loop component. The data is obtained from a JSON structure.
TFPReportCollectionData
a report data loop component. The data is obtained from a TCollection instance. This collection is only available at runtime.
TFPReportObjectListData
a report data loop component. The data is obtained from a TObjectList instance.
TFPReportExportHTML
a report renderer that creates HTML pages.
TFPReportExportfpImage
a report renderer that creates image files.
TFPReportExportPDF
a report renderer that creates PDF files.
TFPreportPreviewExport
a report renderer that previews the report on screen.
TFPreportPrinterExport
a report renderer that sends the report to the printer.
How to use these components ? You need at least 3 components:

1. A report component. (`TFPReport or TFPJSONReport`).
2. A data loop component.
3. A renderer (export) to generate output.

For the report component, it is important to decide in advance whether or not the end user should be able to modify (*to a lesser or larger degree*) the report design. The choice of report component depends on it: When the report design is stored solely in the `.lfm` file, using a `TFPJSONReport` component, the user cannot modify it. When the report design is stored in a file (*to be shipped with the application*), a simple **TFPReport** component can be used. Both components can of course load a design from file, so when in doubt, use a `TFPJSONReport` component.
A data loop component is needed when designing the report in the IDE: the component editor will not allow you to fetch arbitrary data. The data component must be present on the form or data module that contains the report. A report would not be useful without some output, so a renderer is needed.

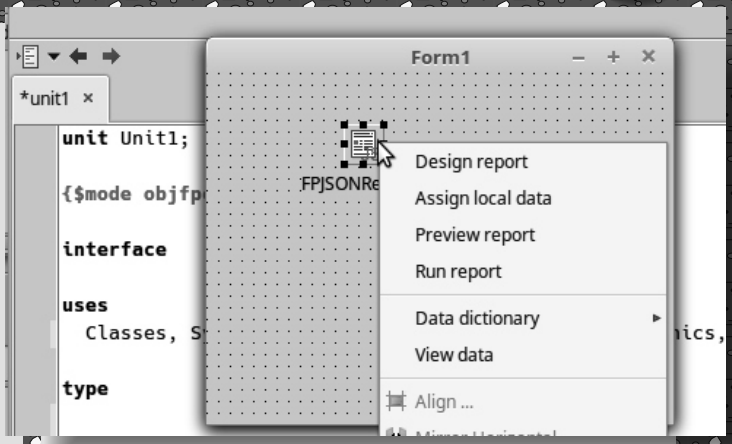


Figure 5 : The report component design menu

To design the report, right-click the report, a component design menu will pop up as show in figure 5. The names of most of these menu items speak for themselves.
The only one that may not be immediately obvious is the 'Assign local data' menu item: When you have three report components and a bunch of data loops on a form or datamodule, you don't necessarily want all data loops to be available to all reports.
The available data loops for a report can be controlled in the Object Inspector: the ReportData property is a collection which enumerates the available data loops for a report instance.
In the simple case where all data loops on the form should be available to the report, the 'Assign local data' item can be used: Clicking this menu item will check the form for data loop components, and will make them available to the report.
Once the report is designed, a small amount of code is needed to show the report. Assuming we've dropped a PDF report export component on the form and configured it properly, the following code will export the report:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  FPJSONReport1.RunReport;
  FPJSONReport1.RenderReport(FPReportExportPDF1);
end;
```

It's clear that writing only these two lines of code is a lot easier than designing and running the report completely in code.

Conclusion

In this article we've shown how to load and save a report design from and to file. We've also shown how grouping and aggregates work. Finally, we've shown that the coding involved in creating a report design can be dispensed with, the visual report designer makes this all a lot easier. In the next article, we'll show how to allow the end-user to design the report, and we'll have a look at some of the more exotic reporting elements in the report engine.

Introduction.
In the last article we spoke about installing Mint on Virtual box including Lazarus. For **OpenSuze** there are quite some differences. This is a bigger environment and has not that easy way of installing things, almost as if you were under Windows.
There are four major steps:

1. Make sure your Virtual Box is correctly installed, you can find that in issue 65.
2. The creation of the Virtual Hard disk for this project.
3. Install OpenSuze as OS.
4. Install Lazarus on OpenSuze.

Because of having to little space in this issue to show all the elements for handling this project we have split it into two parts: Here is the first part. It contains point 1, 2 and 3.
Before you can start creating the virtual hard disk you really need to organize a few things first:
Here is what to do: download the ISO from the OpenSuze website: `https://www.opensuse.org/` probably you will have to set the language of the website, the choice is down right at the bottom.
There are two choices for the version you might want to use:**Tumbleweed or Leap**

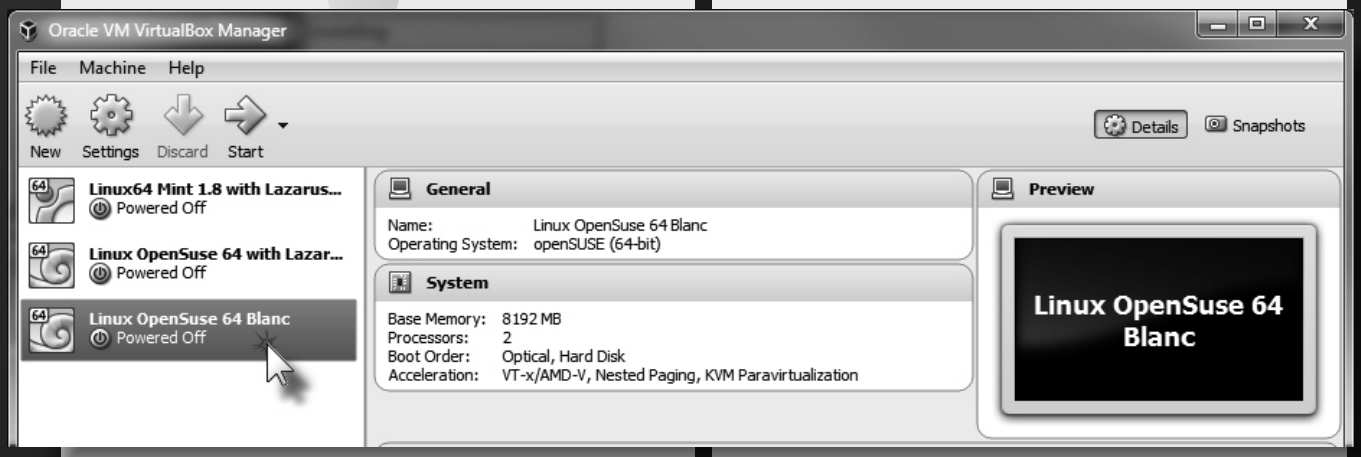
openSUSE Tumbleweed
is a rolling-release. This means the software is always the latest stable versions available from the openSUSE Project. Things will change regularly as Free and Open Source projects continually release new versions of their software. **Tumbleweed** is recommended for Developers.

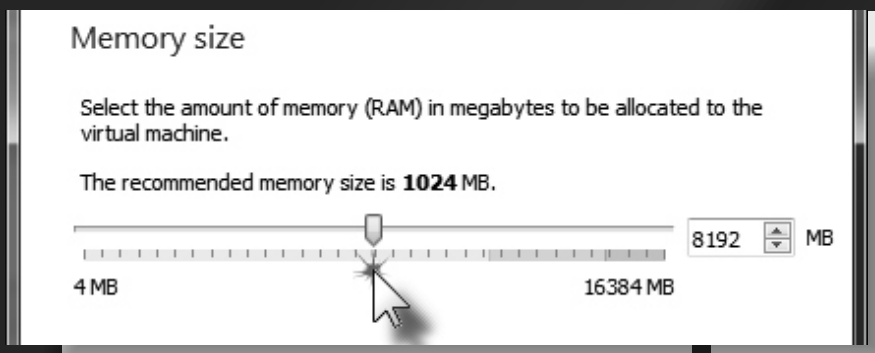
openSUSE Leap
is a regular-release. This means **it releases annually**, with security and stability updates being the priority during each release lifetime. It is not expected to change in any significant way until its next annual release.

Leap shares a Common Base System with **SUSE Linux Enterprise**, so major architectural changes are not expected for several years, aligned with each new Major Release (eg 12, 13, etc) of **SUSE Linux Enterprise.**
Leap is recommended for **Sysadmins, Enterprise Developers, and 'Regular' Desktop Users.** We chose Leap.
The size of the Hard disk is an open question.
You best choose 40 GB because that is sufficient to really make maximum use of the installation.

Which Media to Download
The DVD/USB Stick is typically recommended as it contains most of the packages available in the distribution and does not require a network connection during the installation.
The Network CD/USB Stick is recommended for users who have limited bandwidth on their internet connections, as it will only download the packages they choose to install, which is likely to be significantly less than 4.7GB.

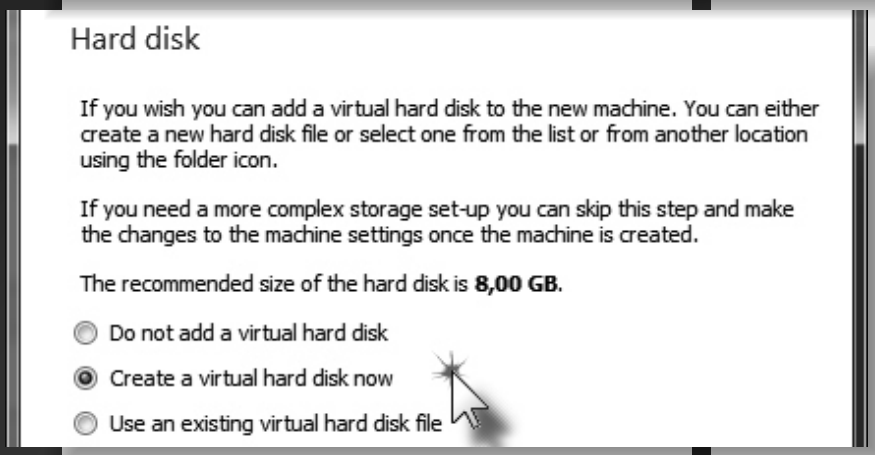
I suggest you arrange for the burning of the DVD since it is an ISO. You could use the program **"Alcohol"** - it has a free version, or some program alike. They are capable of installing a virtual drive. So it won't be necessary to burn it.
Its elementary you have downloaded the ISO and prepared it because you will need that for the installer on **VirtualBox.**
The figures we created will show you the adjustments you should make...



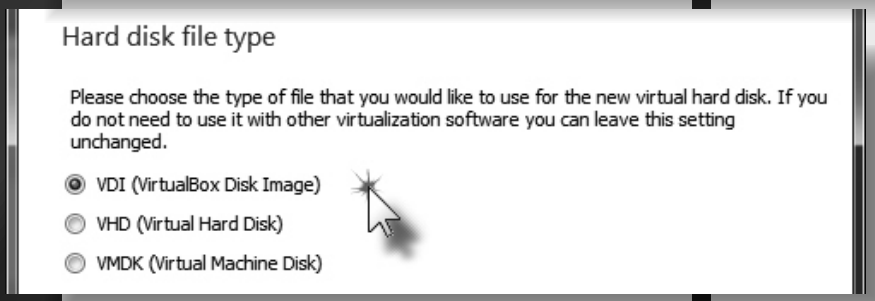


Chose New and type a name.
It can be anything you want.
If you say something like:
Linux OpenSuze it comes automatically with suggestions, but you do not have follow that.

Memory comes up first:
give it a lot but not all. Just follow the next buttons.

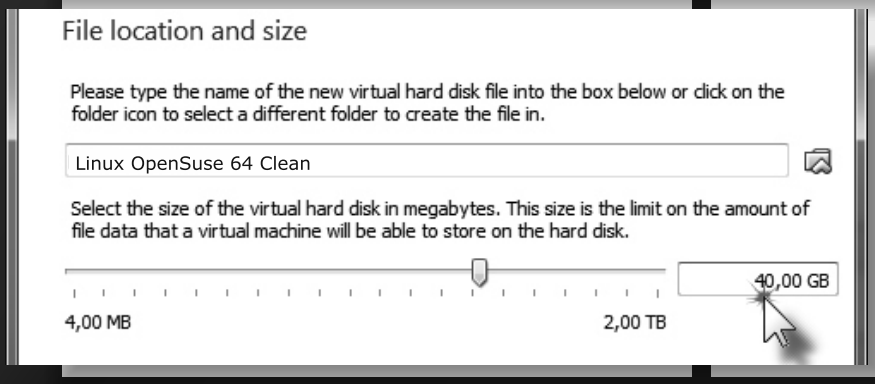


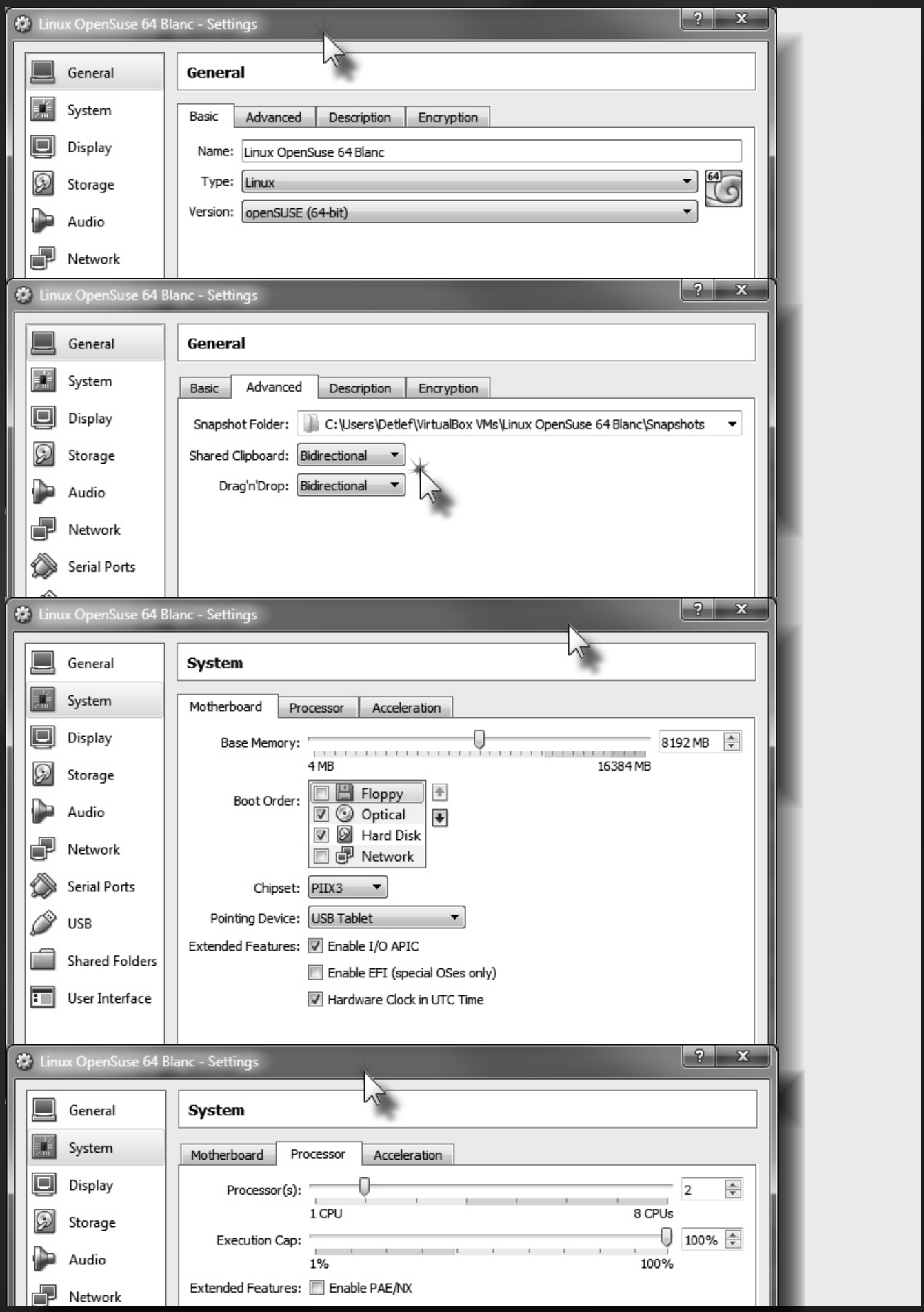
About creating the Hard disk.
You probably better follow the suggestion the program gives you.
Unless of course you know what your doing.
The size and dynamically allocation is very important.
Your best choice is probably to use **virtual disk Image(VDI)**.

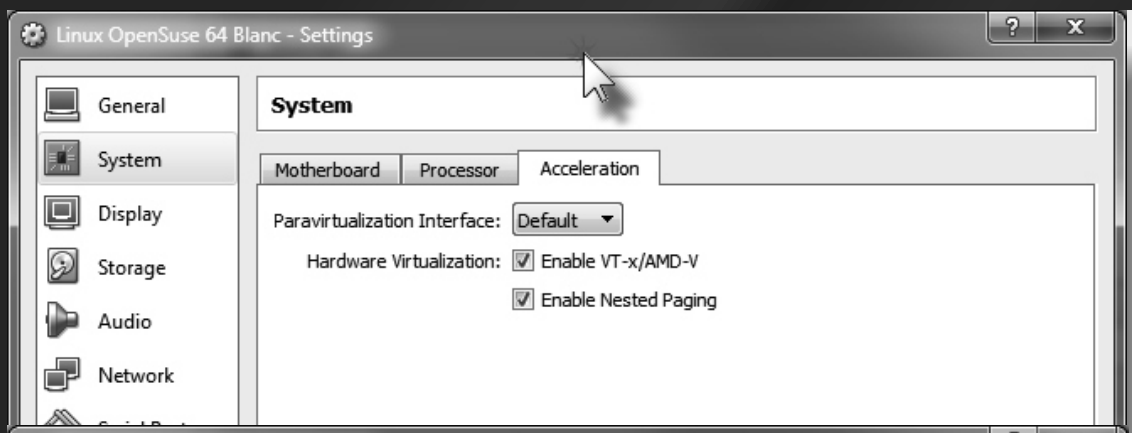


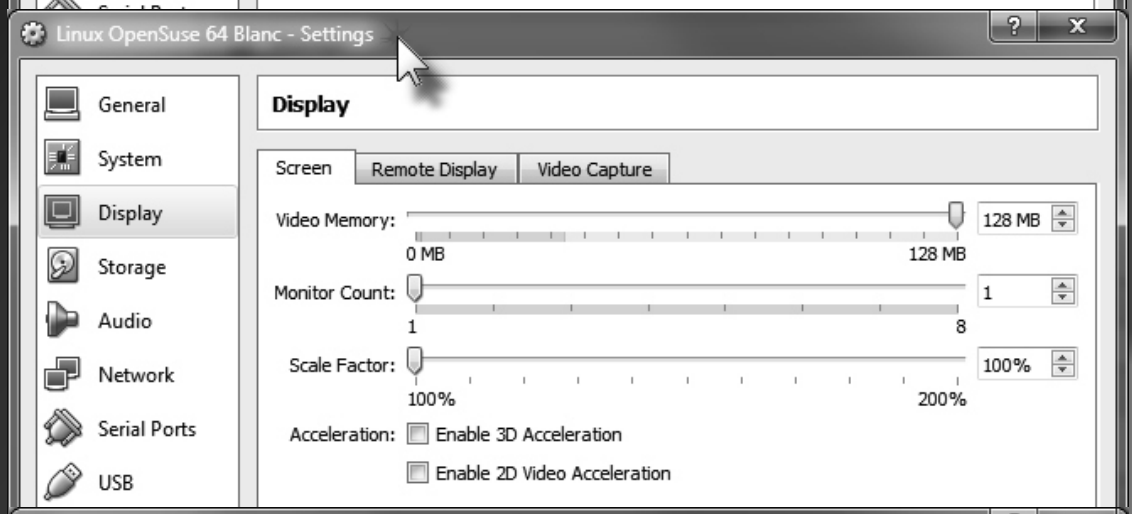
The size of the Hard disk is an open question. You best choose 40 GB because that is sufficient to really make maximum use of the installation.

After you have made the installment you bets go to settings and arrange for the details.
I have made some choice which you probably would like to have as well. See the next pages...

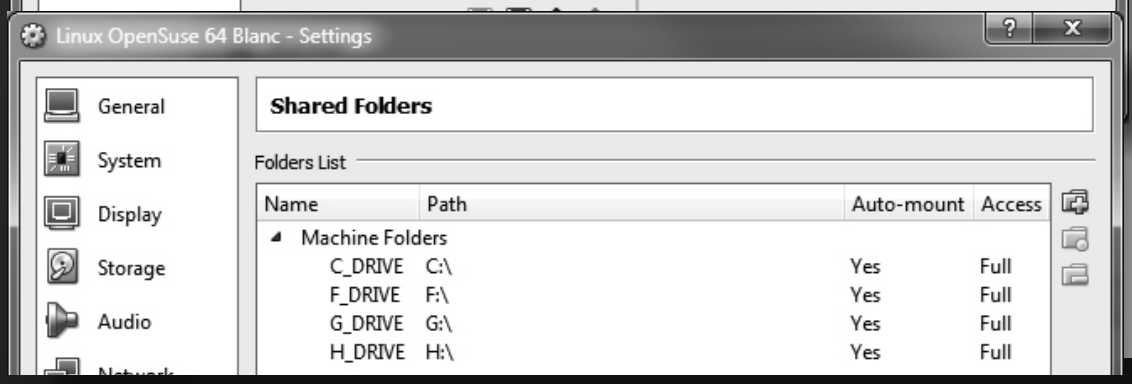


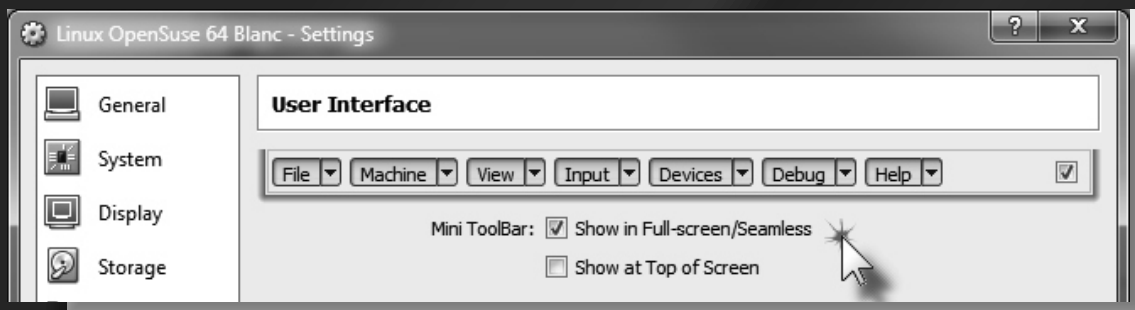

Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Network
General
Basic
Advanced
Description
Encryption
Name: Linux OpenSuse 64 Blanc
Type: Linux
Version: openSUSE (64-bit)
Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
General
Basic
Advanced
Description
Encryption
Snapshot Folder: C:\Users\Detlef\VirtualBox VMs\Linux OpenSuse 64 Blanc\Snapshots
Shared Clipboard: Bidirectional
Drag'n'Drop: Bidirectional
Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface
System
Motherboard
Processor
Acceleration
Base Memory: 8192 MB
4 MB
16384 MB
Boot Order:
Floppy
Optical
Hard Disk
Network
Chipset: PIIX3
Pointing Device: USB Tablet
Extended Features: Enable I/O APIC
Enable EFI (special OSes only)
Hardware Clock in UTC Time
Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Network
System
Motherboard
Processor
Acceleration
Processor(s): 2
1 CPU
8 CPUs
Execution Cap: 100%
1%
100%
Extended Features: Enable PAE/NX


Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Network
System
Motherboard
Processor
Acceleration
Paravirtualization Interface:
Default
Hardware Virtualization:
Enable VT-x/AMD-V
Enable Nested Paging

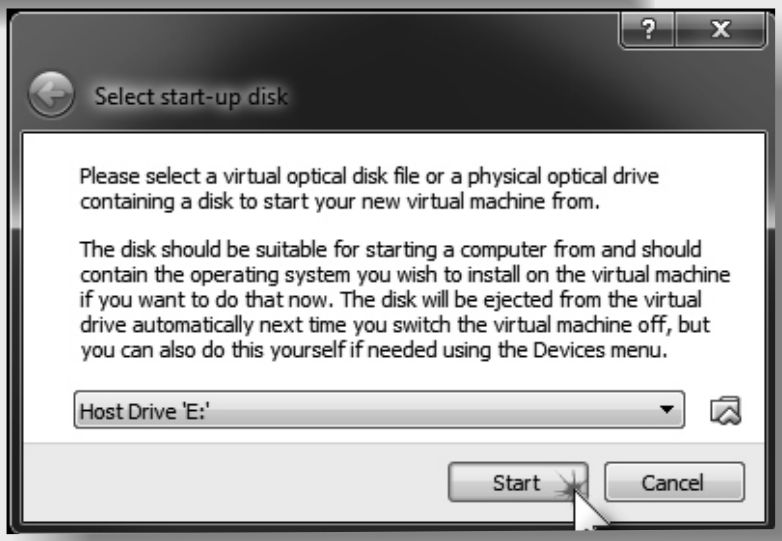

Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Display
Screen
Remote Display
Video Capture
Video Memory:
128 MB
0 MB
128 MB
Monitor Count:
1
1
8
Scale Factor:
100%
100%
200%
Acceleration:
Enable 3D Acceleration
Enable 2D Video Acceleration


Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface
Storage
Storage Tree
Controller: IDE
VBoxGuestAdditions.iso
Controller: SATA
Linux OpenSuse 64 Blanc.vdi
Attributes
Optical Drive:
IDE Primary Slave
Live CD/DVD
Information
Type: Image
Size: 56,74 MB
Location: C:\Program Files\Oracle\VirtualBo...
Attached to: Linux OpenSuse 64 with Lazarus 1..

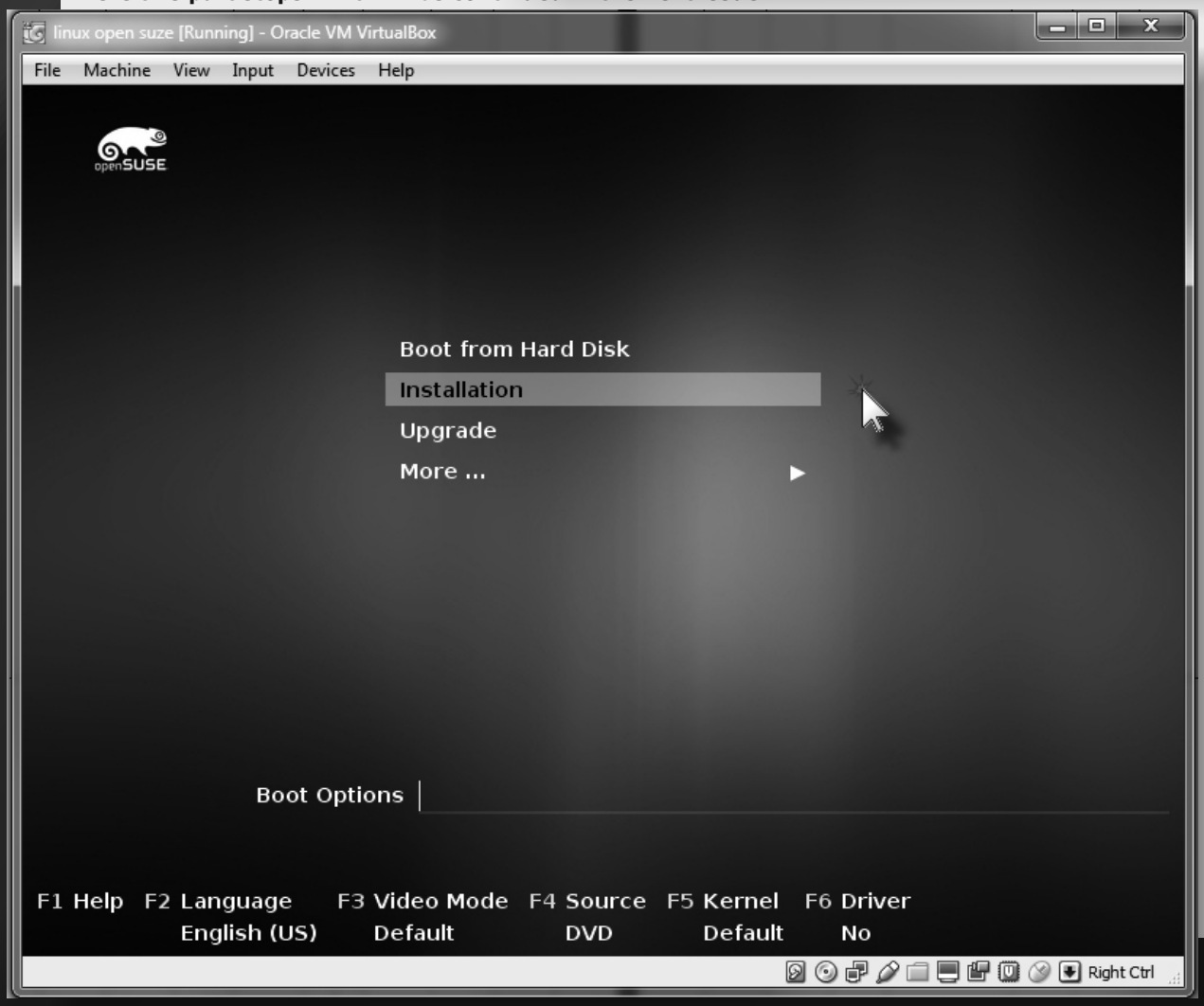

Linux OpenSuse 64 Blanc - Settings
General
System
Display
Storage
Audio
Shared Folders
Folders List
Name
Path
Auto-mount
Access
Machine Folders
C_DRIVE C:\ Yes Full
F_DRIVE F:\ Yes Full
G_DRIVE G:\ Yes Full
H_DRIVE H:\ Yes Full



So far that was easy. We are now going to install the Os finally. **Linux OpenSuze** will be now available as being powered of. Press **START** or click on the green arrow.
The installation process starts: in the window you can see which drive you must choose that contains the installer files.



Here this part stops .... it will be continued in the next issue.



WORKING WITH TACHART

BY WERNER PAMLER

starter — expert

TAChart is a package for drawing graphs, charts and other diagrams for Lazarus under an LGLP license. It is comparable in features with Delphi's TeeChart package, but differs in some features both internally and in the user-interface. While TeeChart covers many aspects of charting by simple properties, TAChart solves this by separate components (such as chart sources and transformations). This model-kit-like design gives increased flexibility and opportunity for code re-use, however, at the expense of a slightly more complex AP.
The package initially was written by Philippe Martinole and was revised by Luis Rodrigues. Alexander Klenin rewrote and expanded much of the code. Werner Pamler is its current maintainer and writer of this article.

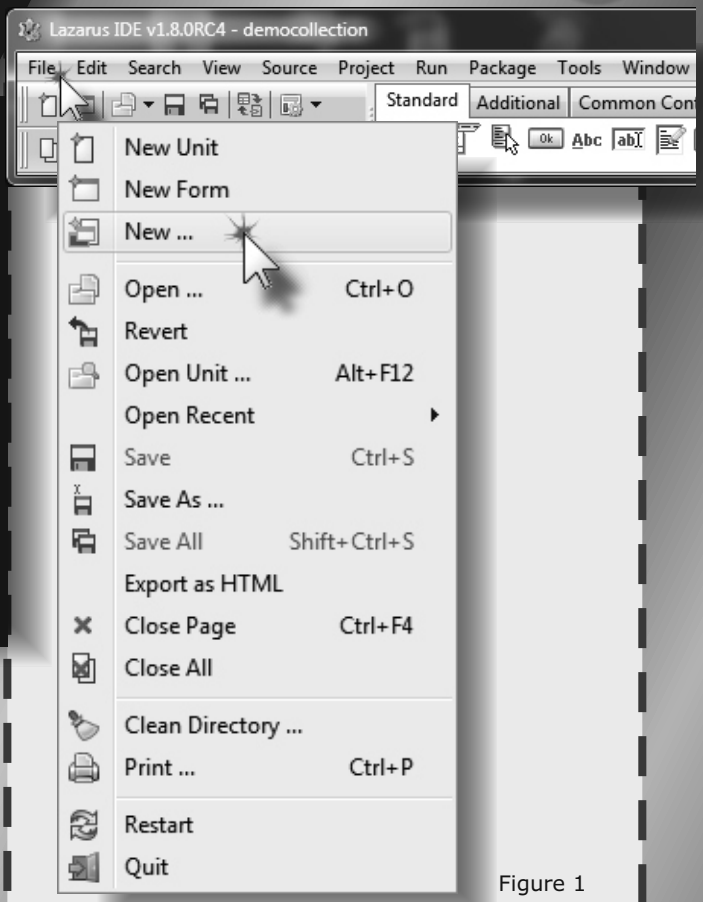


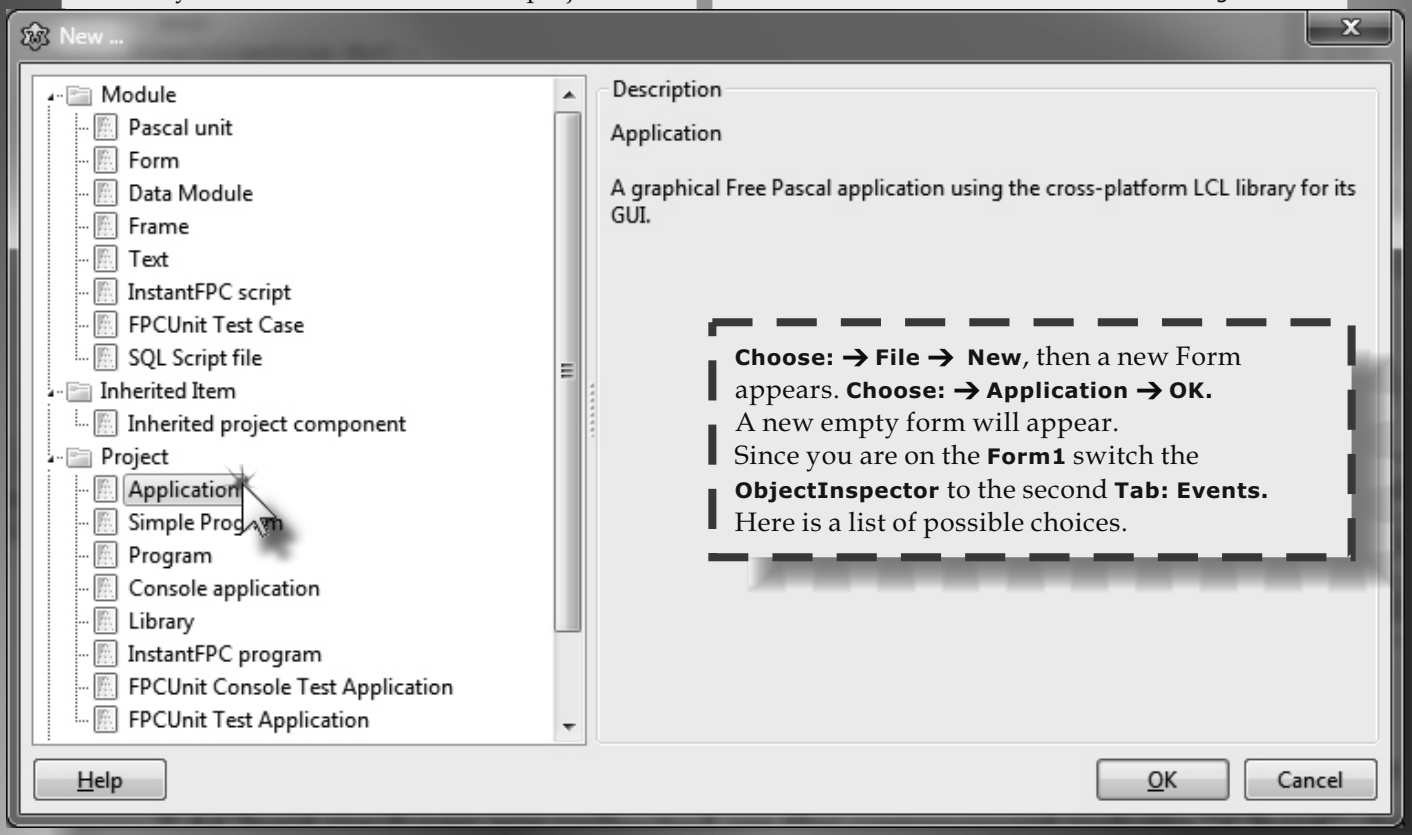
Figure 1

In this introductory chapter the essential steps to create a very simple chart will be demonstrated. We will plot the average monthly temperature in two European cities, Helsinki and Rome. These data, among others, can be found in Wikipedia (`https://en.wikipedia.org/wiki/Climate_of_Europe#Temperature`) and are reproduced here:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Helsinki	-1.3	-1.9	1.6	7.6	14.4	18.5	21.5	19.8	14.6	9.0	3.7	0.5
Rome	11.9	13.0	15.2	17.7	22.8	26.9	30.5	30.6	26.5	21.4	15.9	12.6

To start you will have to create a new project.

Figure 2



Choose: → File → New, then a new Form appears. **Choose: → Application → OK.**
A new empty form will appear.
Since you are on the **Form1** switch the **ObjectInspector** to the second **Tab: Events.**
Here is a list of possible choices.

```
1   unit Unit1;
.
.   {$mode objfpc}{$H+}
.
5   interface
.
.   uses
.     Classes, SysUtils, FileUtil, TAGraph, Forms, Controls, Graphics, Dialogs;
.
10  type
.
.     { TForm1 }
.
.     TForm1 = class(TForm)
15      Chart1: TChart;
.       procedure FormCreate(Sender: TObject);
.     private
.
.     public
20
.     end;
.
.   var
.     Form1: TForm1;
25
.   implementation
.
.   {$R *.lfm}
.
30  { TForm1 }
.
.   procedure TForm1.FormCreate(Sender: TObject);
.   begin
34
35  end;
.
.   end.
38
```

Go for the **OnCreate** event.
Double click and the form will open.
Complete with the code.

Figure 3

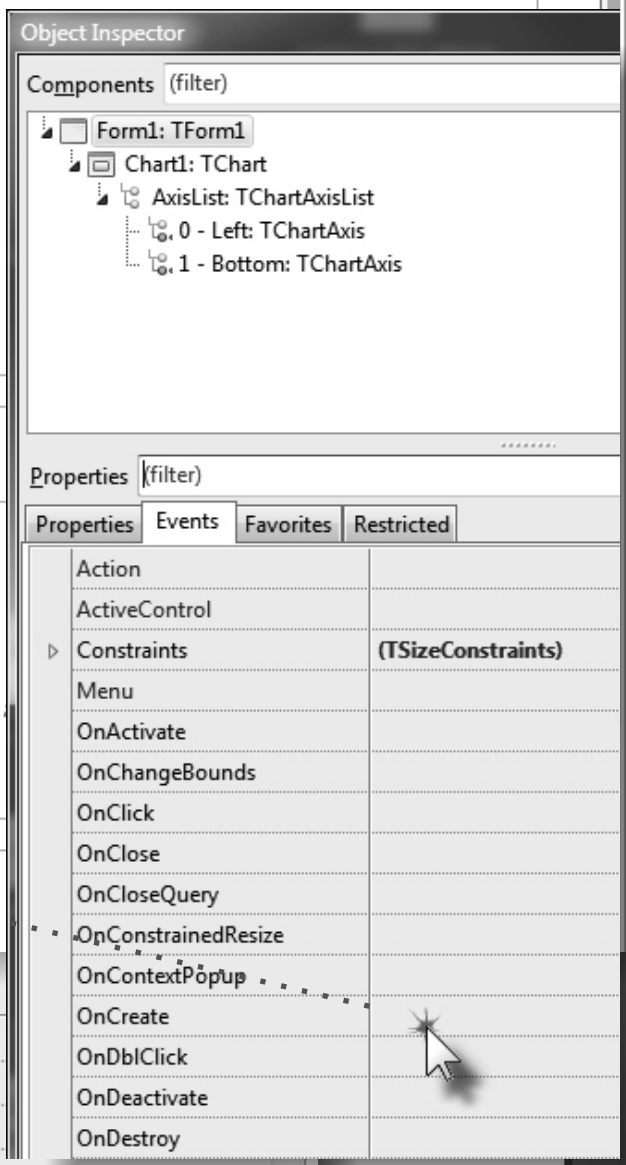


Figure 4

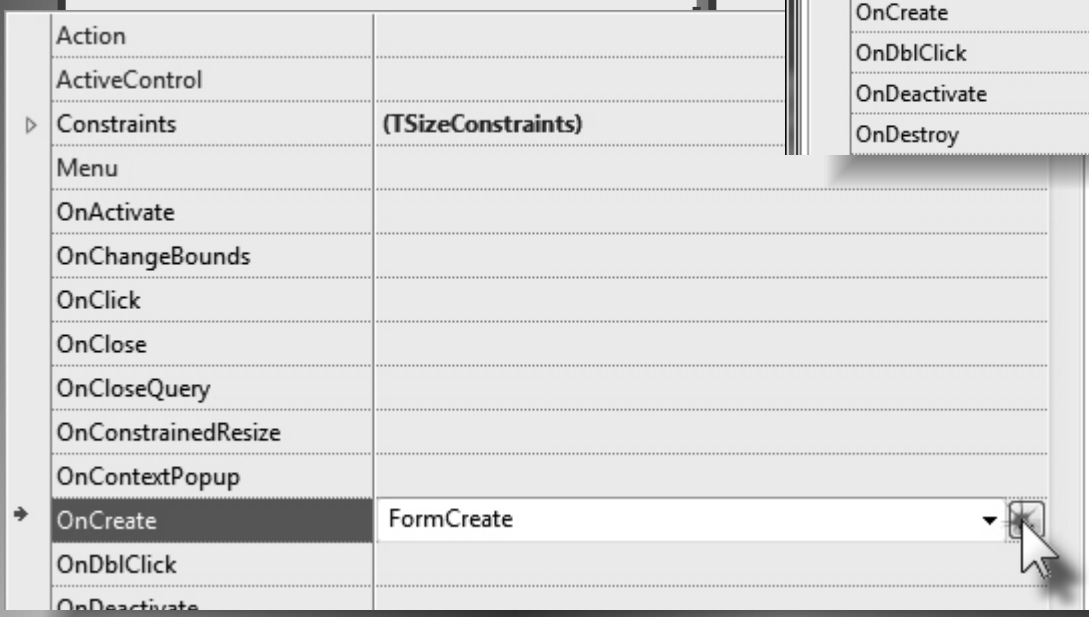


Figure 5

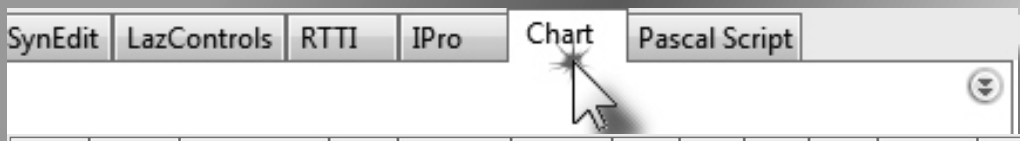


Figure 6

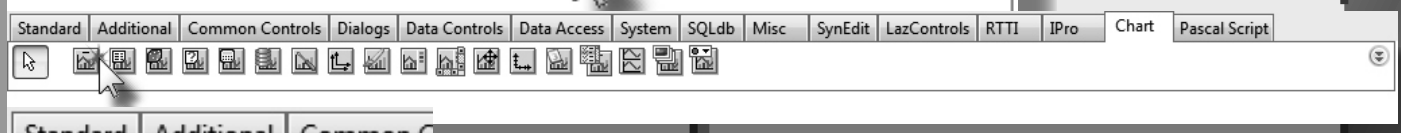


Figure 7

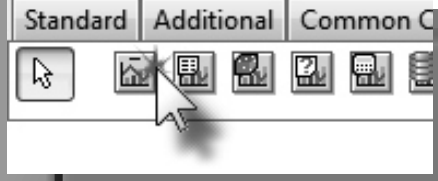


Figure 8

The next step will be moving to the **Chart-Tab** in the component palette. Choose the first component at the left, which is **`Tchart`**. Double-click on it and the component will automatically be placed on the form. You can as well drag and drop the component to the form. Now you can see in the Object Inspector (*Property settings*) the properties of the chart. If you don't, just click on the chart in the **form designer** to select it and bring its properties into the **ObjectInspector.** Find the property "Align" and select the value "alClient" from the dropdown list -- this automatically increases the chart to fill the entire form. Next, maybe rename the chart. By default, it is named **"Chart1",** but it may be more descriptive to name it **"TemperatureChart"** -- find the property `Name` and type the new name in the Name's edit field.".

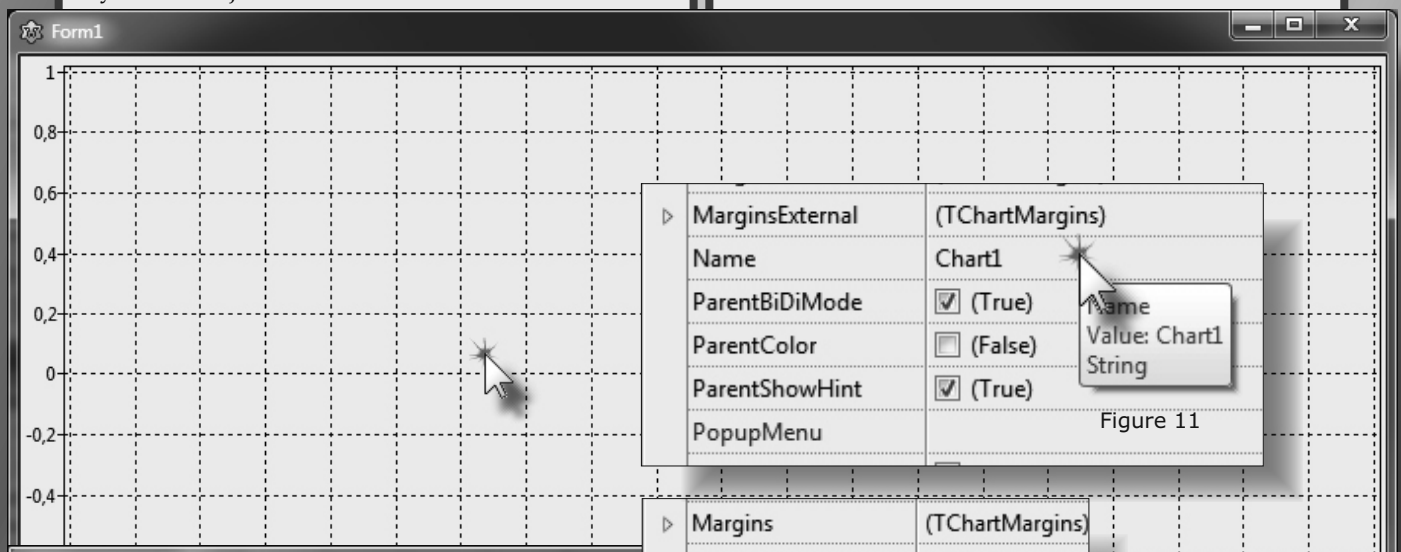


Figure 10

Figure 11

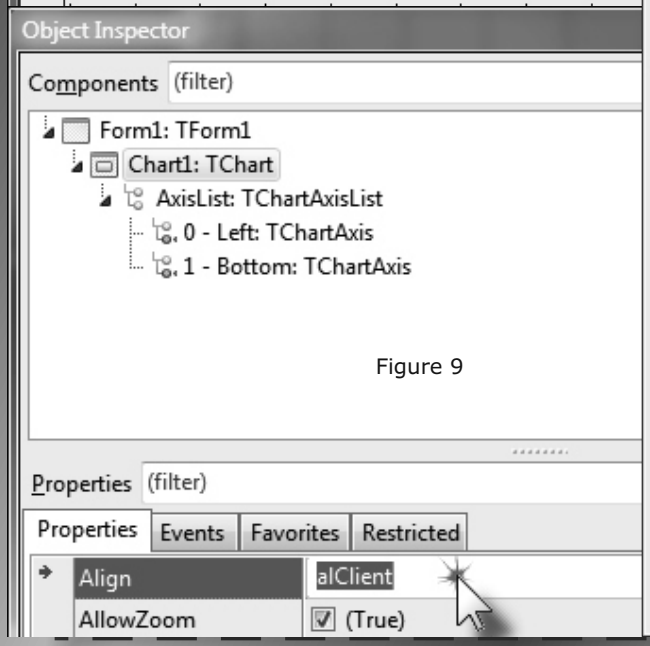


Figure 9

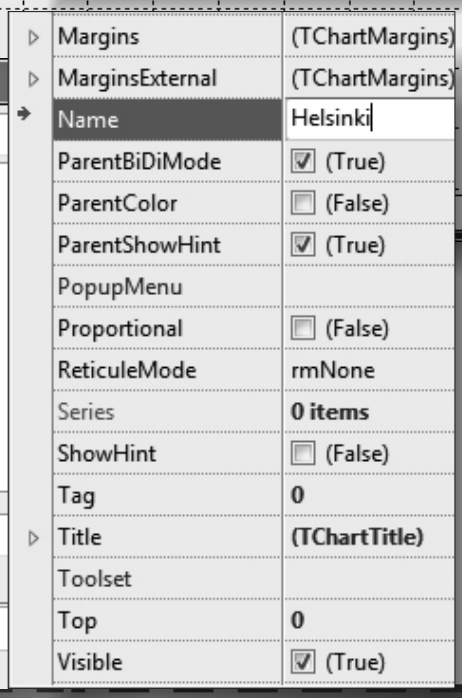


Figure 12

So far, the chart is empty and shows default axes. In **TAChart**, data are displayed by so-called "series"; this naming follows the convention of **Delphi's TeeChart**.
Several series can be overlaid within the same chart. **TAChart** supports many types of series, they will be discussed in detail in other articles. Time-dependent data are often displayed as **"line series"**. This series-type connects data points by linear segments; optional markers can be drawn at each data point.

Figure 13: The series editor of TAChart ready to create a LineSeries.

In order to add a series, double-click on the chart (*or right-click and select "Edit series" from the context menu*). This opens the series editor of **TAChart** from which the requested series type can be selected after clicking the "Add" button (*Figure 13*). Afterwards, the series is selected in the **ObjectInspector.** How can you tell that the series is selected? Look at the **object tree** above the **ObjectInspector. It shows a hierarchical view of all objects on the form**. You see the chart (*named* "TemperatureChart: TChart") with its axes as children, and the new series. It is named by default as " TemperatureChartLineSeries1". This tree node must be highlighted. If not, click on this node to select the series. This is the key step to bring the properties of the series into the object inspector. Now you can change the name of the series to be more descriptive, such as "HelsinkiSeries"

There are several ways to assign data to a series, they will be described in a later article **"Data sources for plotting"**.
The easiest way, and most often used in practice, is to add data by source code. Since we know the data from the beginning we can hook into the **"OnCreate"** = (FormCreate) event of the form and prepare the plot data here by adding the following code to the event handler:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   HelsinkiSeries.Add(-1.3, 'Jan');
   HelsinkiSeries.Add(-1.9, 'Feb');
   HelsinkiSeries.Add( 1.6, 'Mar');
   HelsinkiSeries.Add( 7.6, 'Apr');
   HelsinkiSeries.Add(14.4, 'May');
   HelsinkiSeries.Add(18.5, 'Jun');
   HelsinkiSeries.Add(21.5, 'Jul');
   HelsinkiSeries.Add(19.8, 'Aug');
   HelsinkiSeries.Add(15.6, 'Sep');
   HelsinkiSeries.Add( 9.0, 'Oct');
   HelsinkiSeries.Add( 3.7, 'Nov');
   HelsinkiSeries.Add( 0.5, 'Dec');
end;
```

Each call of the method "Add" inserts a new data point with the **"Y" value** specified by the first parameter to the series.
The second parameter is optional and defines the text by which the point can be labeled on the **X axis**.

NOTE: To show the text on the left axis the Helsinki *(actually* **TChart**) property **Marks** has a style setting which should be set to **smsLabelValue.** Otherwise it will not show!

NOTE that "Add" does not specify a value to be used for the x coordinate – X is just given by the index of the data point, i.e.
The first value (-1.3) will be plotted at x=1,
The second value (-1.9) at x=2, etc.

This works in this particular example in which the data points are equally spaced.
In the more general case of irregularly spaced data the method "AddXY" should be used which requests the X value at the beginning of the parameter list, in addition to those already mentioned. Of course, "AddXY" could be called in our example as well:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 HelsinkiSeries.AddXY( 1, -1.3, 'Jan');
 HelsinkiSeries.AddXY( 2, -1.9, 'Feb');
 [...]
```

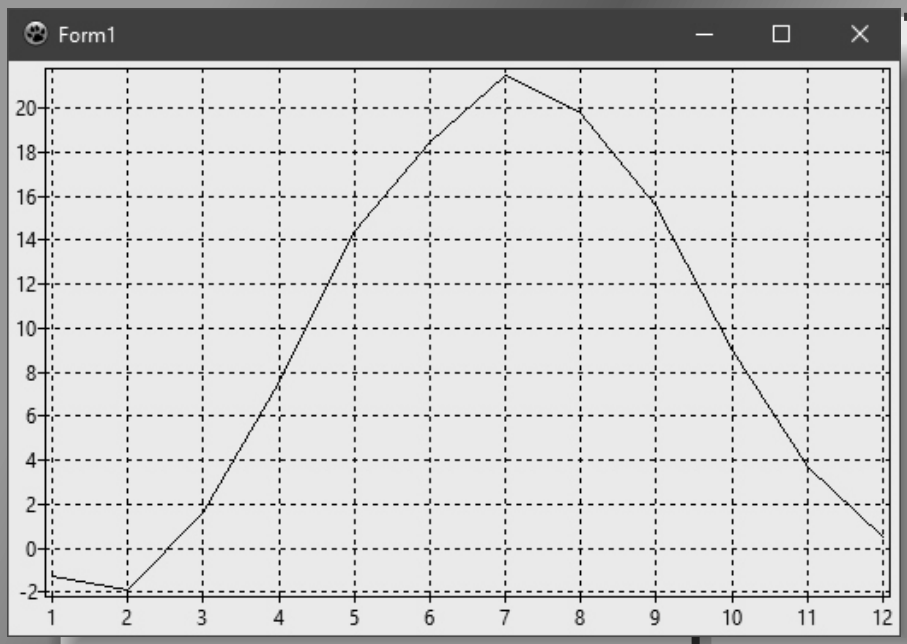


Figure 14: Initial version of the chart. After compilation we see our first chart.
It's not quite perfect, though…

We should give each axis a caption to indicate which quantity is plotted along it:
the x axis should be labeled with "Month", and the y axis with "Average temperature, C".
Due to the complexity of a chart all properties cannot be displayed in the object inspector simultaneously.
It is important to work with the object tree above the **ObjectInspector** to preselect an object for editing.

Underneath the main node
TemperatureChart: Tchart"
we see "AxisList: TChartAxisList"
with two child nodes
"0 – Left: TChartAxis" and
"1 – Bottom: TChartAxis".

Click on the first node to bring the properties of the left axis **(Y)** into the **ObjectInspector** .
Open the sub-properties of "Title" and find "Caption" – this is the text displayed as the axis title;
type the text "Average temperature, C".
Additionally the property "Visible" must be turned on, it is off by default.
Now the axis caption shows up.
Repeat with the X axis. *(See figure 16 next page.)*

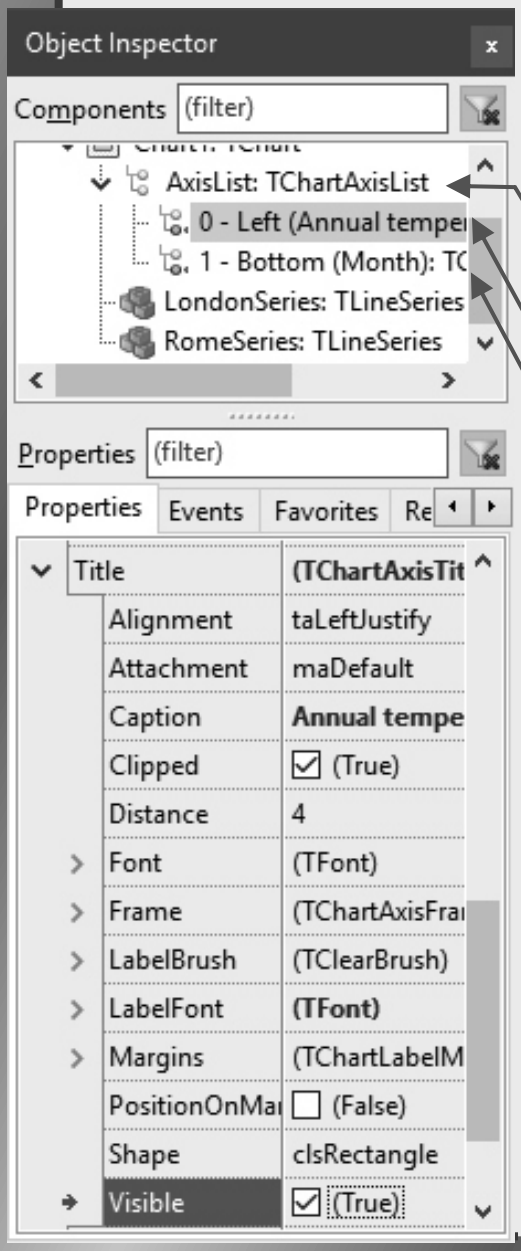


Figure 15: Object tree and object inspector with some properties of the left chart axis.

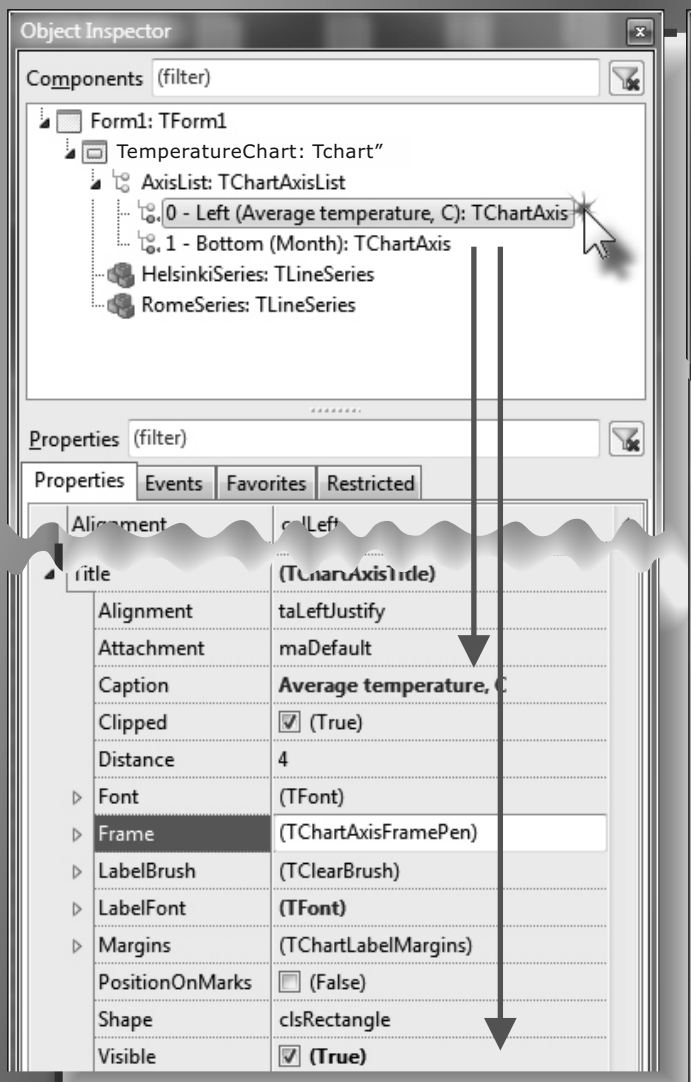


Figure 16: change the caption and visible

Next we fix the **data point axis labels.**
We had entered the month names along with the temperature data – where are they?
They do not show up automatically **because the chart is labeled with numerical data by default**.

Before we can show the axis labels we must understand that the series stores the data to be plotted in a so-called **ListChartSource** which is accessible as property "ListSource" and is a list of records of the x,y coordinates and the data point caption (*along with other data*).

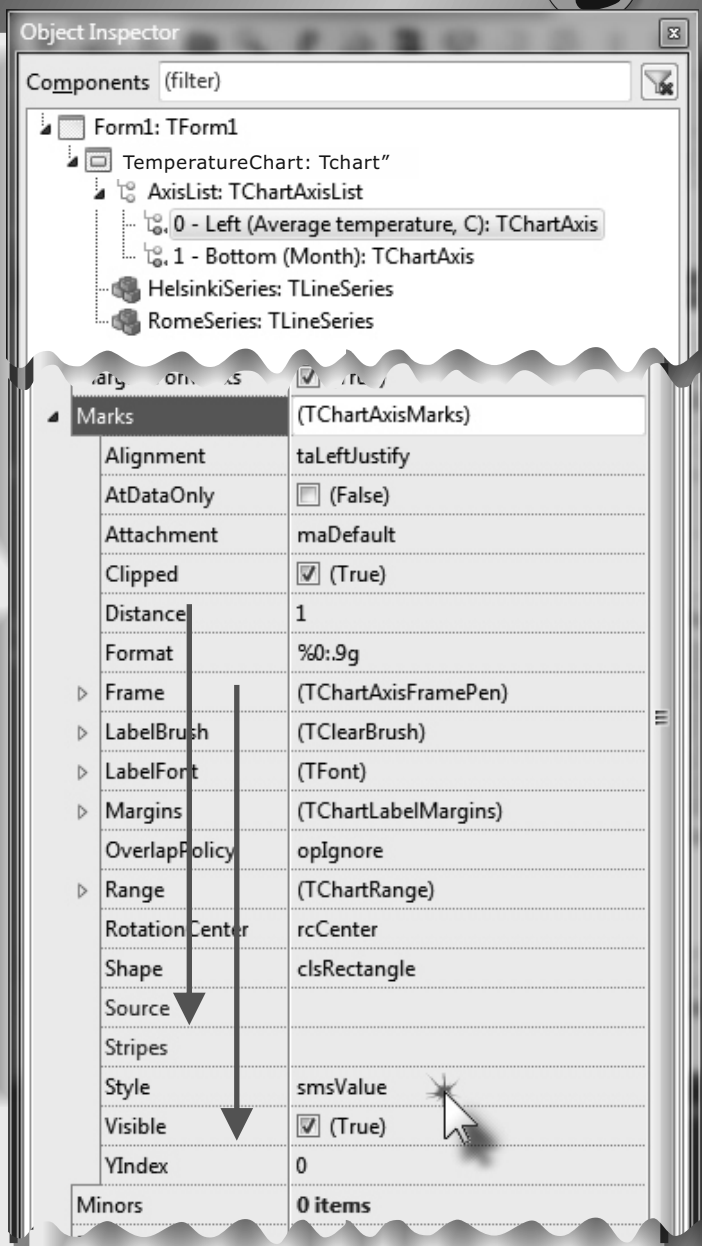


Figure 17: change the caption and visible

The x axis should still be selected from the previous exercise, if not, select it. Open the axis property **"Marks" –** it is responsible for the details of the data tick labels. Among its sub-properties you'll find **"Style"** and **"Source":** "Style" defines which element of the data record will be used for labeling – select the option **"smsLabel"** for the text label. "Source" defines the **ChartSource** from which the labels will be taken. Unfortunately, the ListSource of the series is not accessible in the object inspector, but we can make the assignment in the **`OnCreate`** event which was used already for entering the data:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 [...]
 Helsinki.BottomAxis.Marks.Source :=
            HelsinkiSeries.ListSource;
end;
```

Now that the chart with the first series is finished we can add another series for the temperature data of **Rome.**
Repeat the steps you did to add the Helsinki series, and extend the **FormCreate** method with the **Rome** data. The entire method should now look like this:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 HelsinkiSeries.Add(-1.3, 'Jan');
 HelsinkiSeries.Add(-1.9, 'Feb');
 HelsinkiSeries.Add( 1.6, 'Mar');
 HelsinkiSeries.Add( 7.6, 'Apr');
 HelsinkiSeries.Add(14.4, 'May');
 HelsinkiSeries.Add(18.5, 'Jun');
 HelsinkiSeries.Add(21.5, 'Jul');
 HelsinkiSeries.Add(19.8, 'Aug');
 HelsinkiSeries.Add(15.6, 'Sep');
 HelsinkiSeries.Add( 9.0, 'Oct');
 HelsinkiSeries.Add( 3.7, 'Nov');
 HelsinkiSeries.Add( 0.5, 'Dec');

 RomeSeries.Add(11.9);
 RomeSeries.Add(13.0);
 RomeSeries.Add(15.2);
 RomeSeries.Add(17.7);
 RomeSeries.Add(22.8);
 RomeSeries.Add(26.9);
 RomeSeries.Add(30.5);
 RomeSeries.Add(30.6);
 RomeSeries.Add(26.5);
 RomeSeries.Add(21.4);
 RomeSeries.Add(15.9);
 RomeSeries.Add(12.6);

 Helsinki.BottomAxis.Marks.Source :=
         HelsinkiSeries.ListSource;
end;
```

NOTE that it is not required to repeat the month label for the Rome series. Compile and run to see that there are two curves now.
But which curve belongs to which city?

This can be solved by means of a legend.
Of course, this is built into **TAChart.**
The chart has a property "Legend" – select it in the object inspector. It is only the property "Visible" which has to be turned on to show the legend.
But the legend is drawn only as a box with two line segments in it. It still not clear which curve is corresponds to which city…

This is because we must give each series a title, i.e. the city name. Select each series in the object tree – the series are **underneath** the "Helsinki" node -, find the property "Title", and type the city name. In addition we should give each curve an individual color.

You can do this with the property "SeriesColor"; select **clBlue** for Helsinki (blue for "cold") and **clRed** for Rome (red for "hot").
Now do a final compile to see the *(almost)* finished chart.

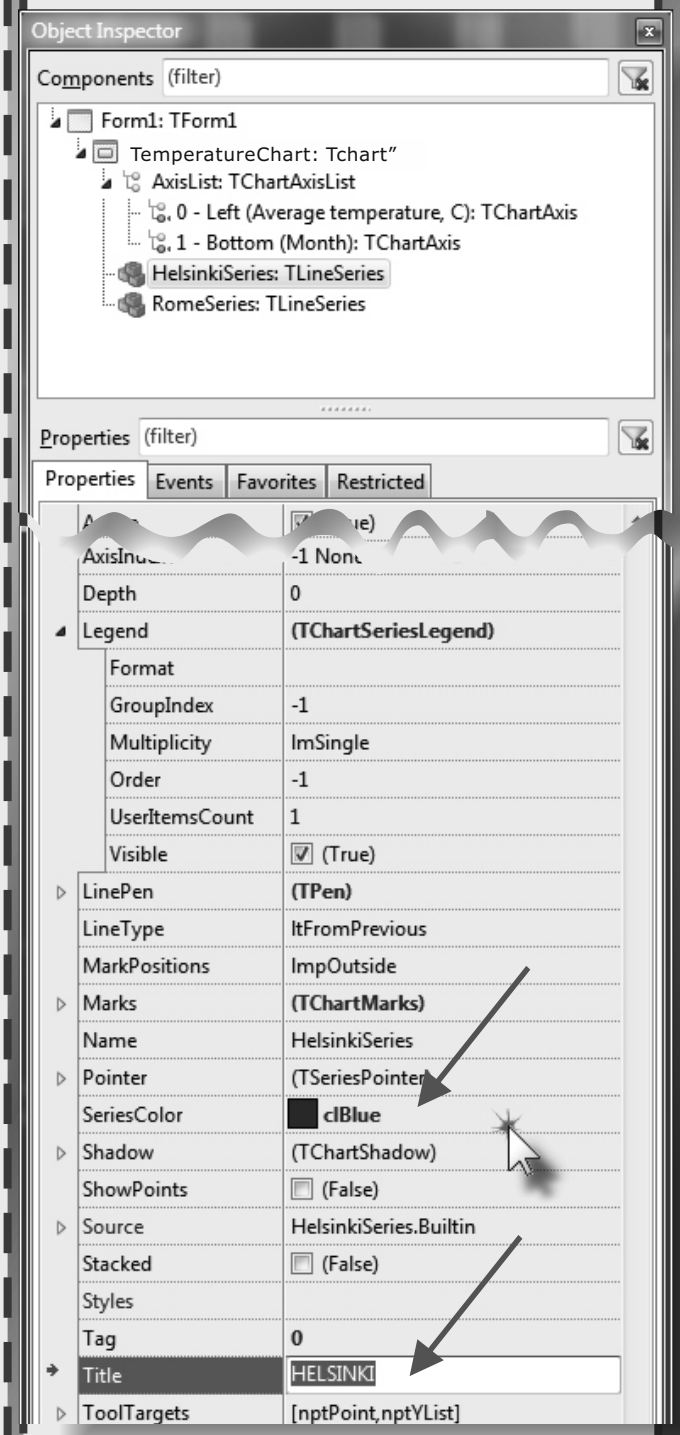


Figure 18: Details of the series

Some final ideas to brush up the chart:

- Add a title above the chart: go to the chart's property **"Title",** find the sub-property **"Text"** and enter "Monthly temperatures In Rome and Helsinki"; check **"Visible"** to show the title. *(See Figure: 19, 20).*
  Set the **"Font.Style"** of the **Title** to **`fsBold`**. *(See Figure: 21)*
- Give the chart area a white **background** color by changing the chart property **"BackColor".** *(See Figure: 22).*
- Show the individual data points of each series by turning on the property **"ShowPoints"** of each series. *(See Figure: 23 on the next page)*
  Set the color of the data points to the color of the connection lines by changing **"Pointer.Brush.Color"** of each series.
- The **legend** is built up in the order in which the series were added to the chart.
  If you want to rearrange the **legend items** to be in the same up-down order as the curves you can check the **property "Inverted"** of the chart's property **"LEGEND".**
  Or you can enter the **(zero-based) index of the legend item** in the sub-property **"Order"** of the series property **"Legend"** *(note that both chart and series have a property "Legend")*.

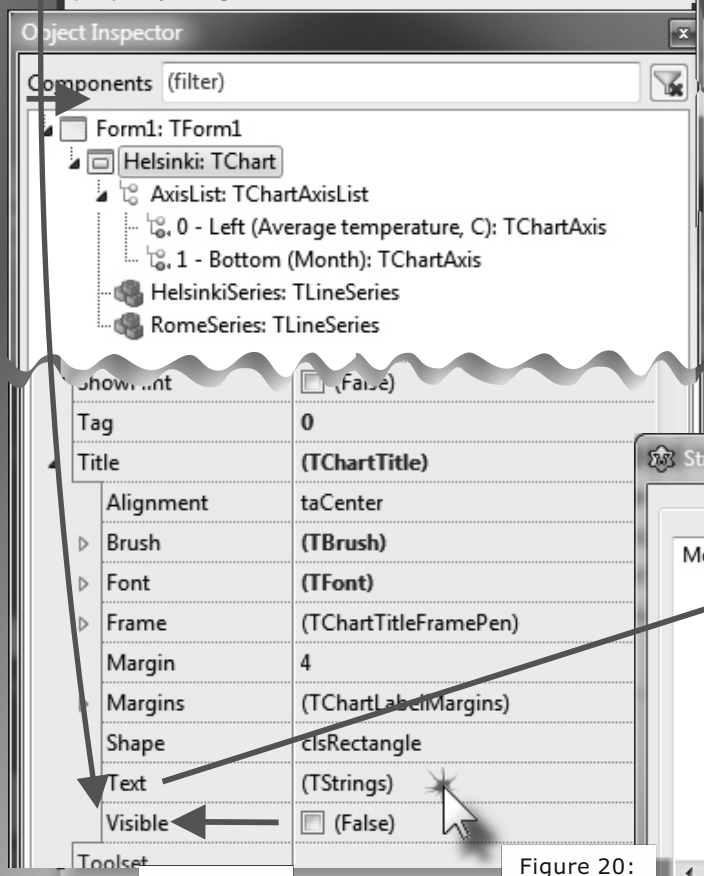


Figure 19:

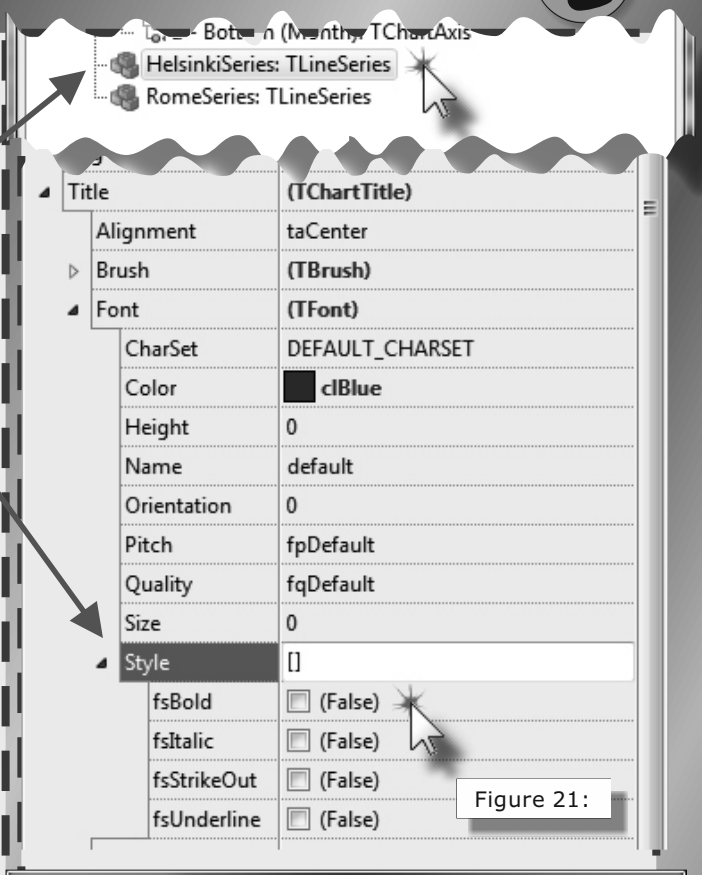


Figure 21:

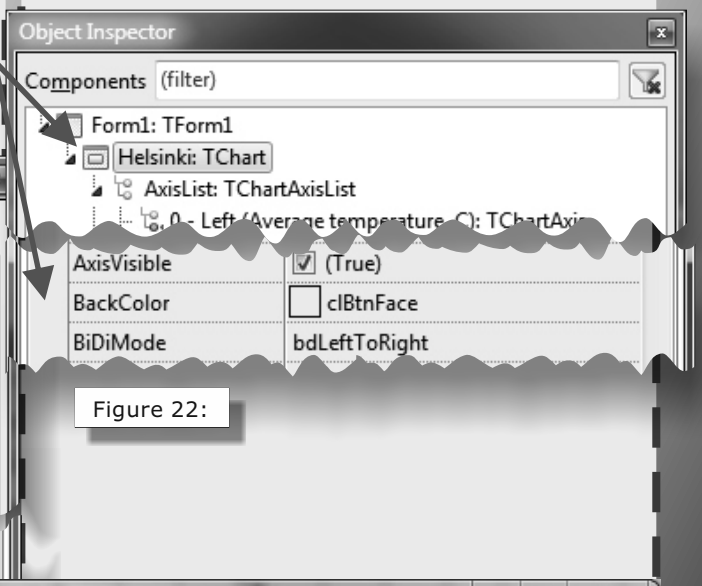


Figure 22:

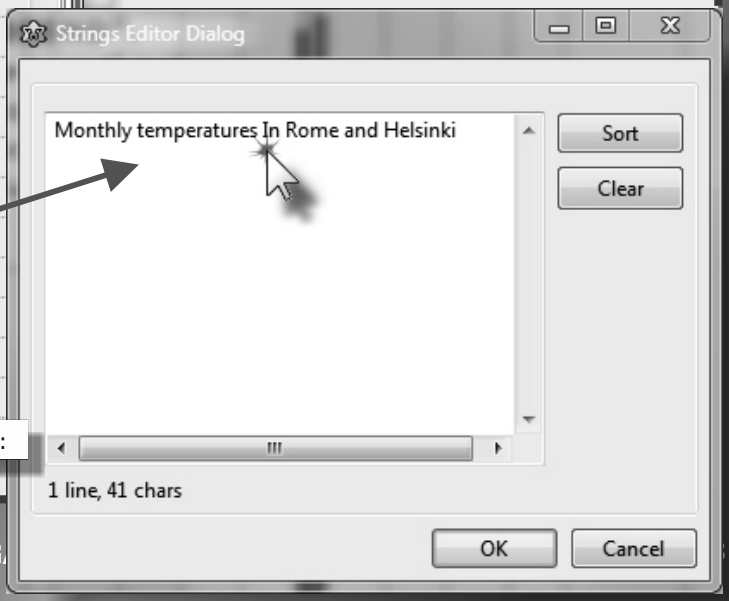


Figure 20:

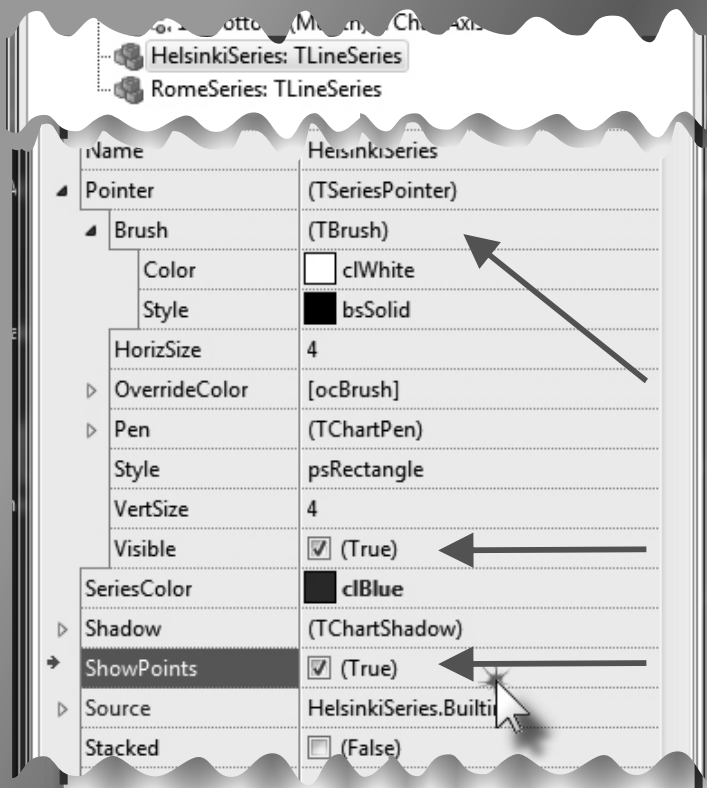


Figure 23: "ShowPoints" of each series. Set the color of the data points to the color of the connection lines by changing "Pointer.Brush.Color" of each series.

INNER STRUCTURE OF A CHART

A chart is a relatively complex object consisting of axes, series, legend, title and footer. Two axes are shown by default, but it is possible to add additional axes at any side of the chart. The axes are administrated by the chart in an "AxisList" (`TChartAxisList`).

The class **`TChartAxis`** from which each axis is instantiated contains a variety of other properties, among them "Title", "Marks" with ticks and labels, "Grid", or "Minors" (*a collection to define minor tick marks*).
While an axis is labeled in "world" units it may use different logical units internally.
A logarithmic axis, for example, may be labeled as **`"1", "10", "100"`** etc, but the axis internally plots the logarithms, **`"0","1","2"`** etc. **`TAChart`** calls the former coordinates "axis" units and the latter "graph" units. In case of non-transformed axes both units are the same.

The graph units are mapped to the pixels on the screen which are called "image" coordinates. These names are important to understand the conversion functions used by **`TAChart`**.
Above the chart, a general title (*property* "Title: TChartTitle") can be displayed; similarly a footer (*property* "Foot: TChartTitle") can be added at the bottom of the chart.
A legend (*property* "Legend: TChartLegend") is built-in to help identifying the series or data points.
The series, finally, are the curves plotted within the chart. They are collected by the list "Series" (class **`TCharSeriesList`**) of the chart.
The series are highly **polymorphic** objects.
The element **`Series[index]`** returns only the most basic series type, **`TBasicChartSeries`**.

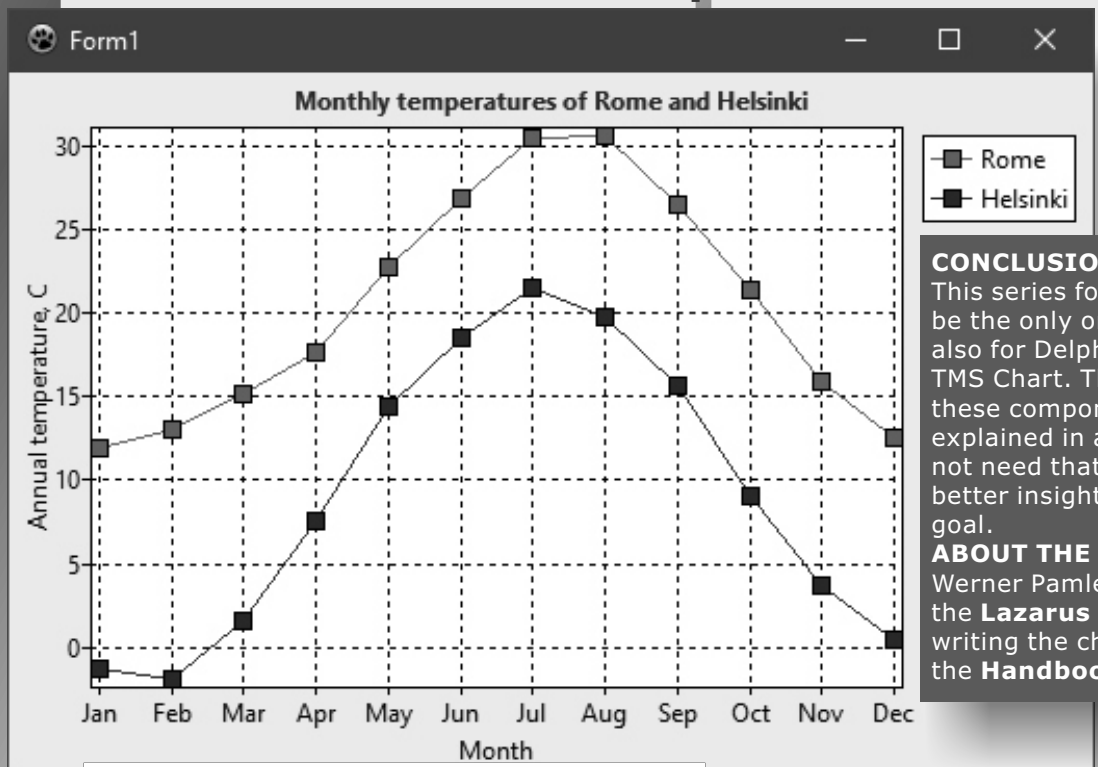


Figure 24: Completed chart of the introductory tutorial

CONCLUSION:
This series for Lazarus will not be the only one. We will do this also for Delphi (TChart) and for TMS Chart. The complexity of these components is as we hope explained in a way that you will not need that much time to get a better insight and achieve your goal.

ABOUT THE AUTHOR
Werner Pamler is a volunteer for the **Lazarus Community** who is writing the chapter TAChart in the **Handbook for Lazarus**

The Majorana Demonstrator experiment is looking for a sign that neutrinos are their own antiparticles. A joint Fermilab/SLAC publication