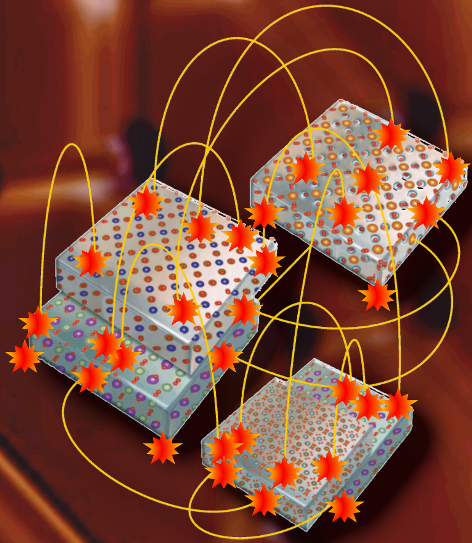
BLAISE PASCAL MAGAZINE 116

Multiplatform / Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js
Databases / CSS Styles / Progressive WebApps
Android / IOS / Mac Windows & Linux

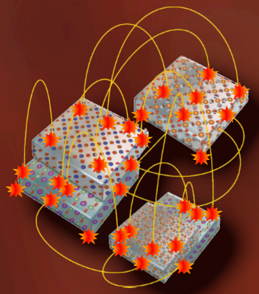


BLAISE PASCAL MAGAZINE 116

Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js / Databases / CSS Styles / Progressive Web Apps Android / IOS / Mac / Windows & Linux

CONTENT

ARTICLES



ADVERTISING

Niklaus Wirth

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

Publisher: PRO PASCAL FOUNDATION in collaboration © Stichting Ondersteuning Programmeertaal Pascal

CONTRIBUTORS

Stephen Ball http://delphiaball.co.uk DelphiABall	Dmitry Boyarintsev dmitry.living @ gmail.com	Michaël Van Canneyt ,michael @ freepascal.org	Marco Cantù www.marcocantu.com marco.cantu @ gmail.com
David Dirkse www.davdata.nl mail: David @ davdata.nl	Benno Evers b.evers @ everscustomtechnology.nl	Bruno Fierens www.tmssoftware.com bruno.fierens @ tmssoftware.com	Holger Flick holger @ flixments.com
Mattias Gärtnernc- gaertnma@netcologne.de	Max Kleiner www.softwareschule.ch max @ kleiner.com	John Kuiper john_kuiper @ kpnmail.nl	Wagner R. Landgraf wagner @ tmssoftware.com
Vsevolod Leonov vsevolod.leonov@mail.ru	Andrea Magni www.andreamagni.eu andrea. magni @ gmail.com www.andreamagni.eu/wp		Helmut Elsner Korrektor der Deutschen Ausgabe helmut.elsner@live.com
		Paul Nauta PLM Solution Architect CyberNautics paul.nauta @ cybernautics.nl	
Kim Madsen www.component4developers.com kbmMW		Boian Mitov mitov @ mitov.com	
	Jeremy North jeremy.north @ gmail.com	Detlef Overbeek - Editor in Chief www.blaisepascal.eu editor @ blaisepascal.eu	
Anton Vogelaar ajv @ vogelaar-electronics.com	Danny Wind dwind @ delphicompany.nl	Jos Wegman Corrector / Analyst	Siegfried Zuhr siegfried @ zuhr.nl

Editor - in - chief
Detlef D. Overbeek, Netherlands Tel.: Mobile: +31 (0)6 21.23.62.68
News and Press Releases email only to editor@blaisepascal.eu

Subscriptions can be taken out online at **www.blaisepascalmagazine.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 or PayPal
Name: Pro Pascal Foundation (Stichting Ondersteuning Programeertaal Pascal)IBAN: NL82 ABNA 0441960863 BICABNANL2A VAT/ NL814254147B01
Subscription department Edelstenenbaan 21 / 3402 XA IJsselstein, Netherlands + 31 (0) 6 21.23.62.68 office@blaisepascal.eu

Trademarks All trademarks used are acknowledged as the property of their respective owners.
Caveat Whilst we endeavor to ensure that what is published in the magazine is correct, we cannot accept responsibility for any errors or omissions.
If you notice something which may be incorrect, please contact the Editor and we will publish a correction where relevant.

Member of the Royal Dutch Library KB KONINKLIJKE BIBLIOTHEEK Member and donor of WIKIPEDIA

SUBSCRIPTIONS (2023 prices)

	Internat. excl. VAT	Internat. incl. 9% VAT	Shipment	TOTAL
Printed Issue (8 per year) **±60 pages :**	**€ 200**	**€ 218**	**€ 130**	**€ 348**
Electronic Download Issue (8 per year) **±60 pages :**	**€ 64,22**	**€ 70**		

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

From your editor

Hello dear readers,
as with this commentary, I begin by expressing a joyous wish for you...: Happy Easter!
The hares will not emerge from your top hat, according to a Dutch saying, but will really surprise you.
I think this issue contains many surprises, some of which are very hopeful:
Read the article on the Open Web set up in Europe. There is a lot going on with it and I think it will be a real success. There is also a lot at stake of course: freedom of speech and our democracy, to which I am personally very attached, and the forecast that politics has to go through an even deeper valley before they realize what they are doing.

I heard with great sadness that a nevertheless so young Mattias Eissing has died.
It is a pity he did not get to live a long life like me. He would certainly have done a lot for the community.
We wish his family and friends strength and send our condolences.

In contrast (feels unfair) it is given to me to become 77 years old and hope to do this work for a very long time to come.
I don't think I have to convince anyone that it is a lot of work to do all the Magazine.
Still, I hope I can get some extra help from you as a reader: we have a great need for practical helpers. For example, people who have a good command of the English language. Or Brazilian (Portuguese).
That mainly for quality purposes. But to Readers and developers who want to write articles we are in VERY great need. It is not anything to be afraid of. Writing an article is often great fun and it also often helps to gain more insight into your own work.

We actually want to set up our own Forum where our readers can interact with each other. For this, of course, another supervisor is needed. So please sign up!
As part of attracting young people, I have set up a section where small very useful code snippets can be displayed.
We need submissions for that too.
I actually also wanted - since I am doing this alone - to also create an editorial board.
This is because we need to create something future-proof and I want to organize my own dissent.
There could be a division of labour. Of course, as a reader, you are also welcome to criticize.
We have something to gain from that.

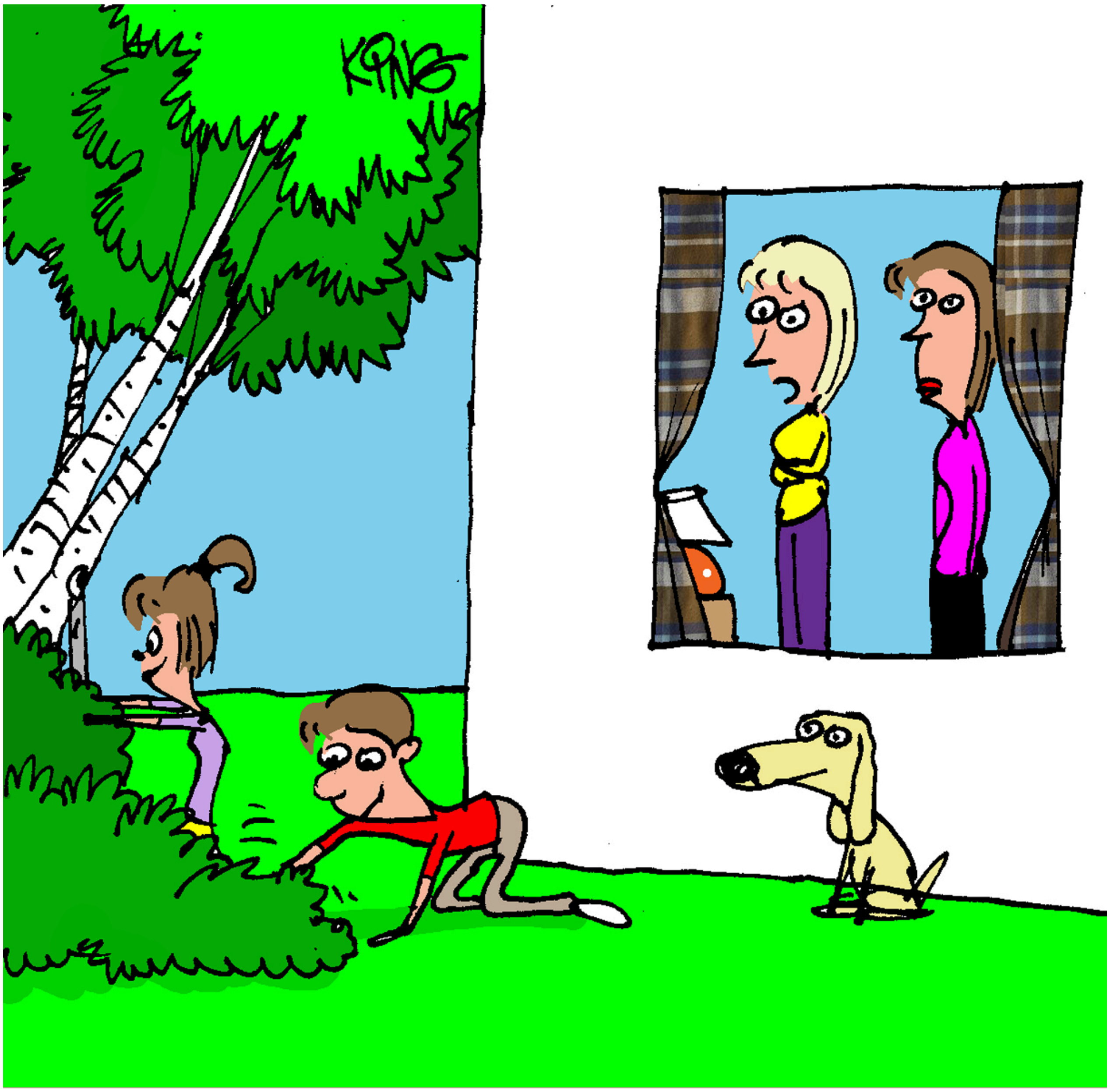
I am currently working very emphatically on a first lesson for Pascal for young people.
And that starts comprehensively and at the very lowest level. Where all the abstract concepts are explained in an understandable way. This is a MUST because we need intake as you all know.
The best language deserves the best youth. I will write a lot more about it....

In this issue you will also find an advertisement about the Summit on 13/14 June. We will also be present there with a stand. Maybe we can get to know each other.

Your Editor:

Detlef

From our technical Advisor, Jerry King



"I had to tell them there were phones in the Easter eggs. It's the only way to get them outside and search for the eggs I hid."

GDK software

embarcadero®
Official Technology Partner

We are GDK.

Do you have a Delphi challenge and are you looking for expertise or capacity? Our experts and developers are ready to realise your ambitions and goals.

30	**Delphi developers** Work with our Delphi Experts
99+	**Delphi conversions** Smart upgrade to the latest Delphi
5	**Embarcadero MVP's** Authority within the Delphi community
4	**Offices worldwide** The Netherlands, UK, Brazil and USA

GDK IN A NUTSHELL

About GDK.

We share a passion for software development and love to keep up with the latest technologies. We have a strong team of specialists with a lot of knowledge and expertise in Delphi.

ACHIEVING YOUR AMBITIONS

Work together.

Looking for a partner to maintain and extend your software? Or upgrade your software to the newest Delphi version? We are ready to help you!

Contact us www.gdksoftware.com

SCAN ME

Codolex
Simplify development

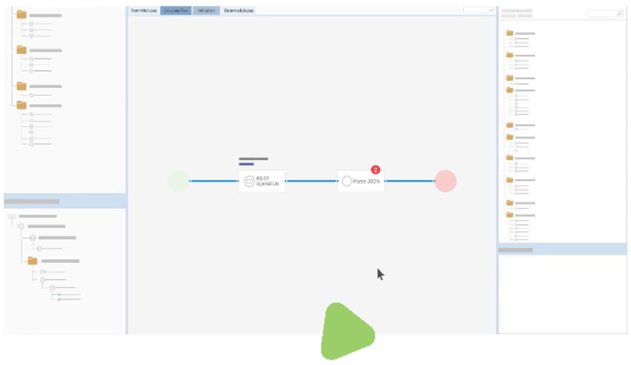
embarcadero®
Official Technology Partner

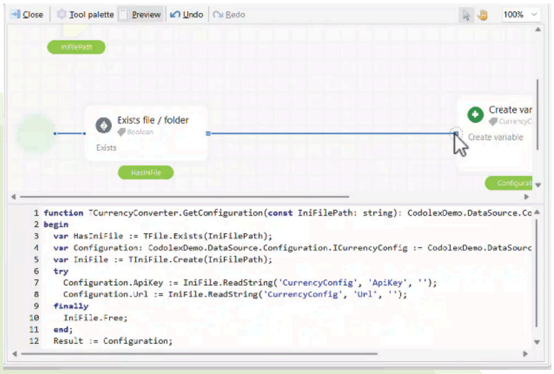
Revolutionary.

Codolex is a low code solution specifically created for Delphi allowing you to develop rapidly whilst remaining in control of the source code.

Visual development

Develop your code through visual design. Understand logic faster through visual representation and adjust easily by modifying the flow. A picture says more than a thousand words!





```
function TCurrencyConverter.GetConfiguration(const IniFilePath: string): CodolexDemo.DataSource.Co
begin
  var HasIniFile := TFile.Exists(IniFilePath);
  var Configuration: CodolexDemo.DataSource.Configuration.ICurrencyConfig := CodolexDemo.DataSourc
  var IniFile := TIniFile.Create(IniFilePath);
  try
    Configuration.ApiKey := IniFile.ReadString('CurrencyConfig', 'ApiKey', '');
    Configuration.Url := IniFile.ReadString('CurrencyConfig', 'Url', '');
  finally
    IniFile.Free;
  end;
  Result := Configuration;
```

Code generation

Codolex generates code that is immediately usable in your projects. Using the preview function in the modeller you can immediately see what code is generated for the flow. There is no vendor lock-in because everything is generated into code.

Codolex provides everything to simplify your development!

SCAN ME

www.codolex.com

Try it out

MATTHIAS EISSING †

It is with deep sadness that we bid farewell to our long-time colleague and friend Matthias Eißing, who passed away suddenly and unexpectedly on 14 February 2024 at the age of 53. We are were all extremely shocked by the news of his death. Matthias leaves behind his partner of many years, Maria, and all of us, in profound mourning.

Many knew Matthias as a very competent and technically adept colleague who now leaves a huge gap. His great love of cats, which he not only adored at home but also liked to incorporate into his presentations, remained a secret known only to a select few. Matthias was the leading presenter at all German live and online events and his knowledge, skills, and support were always greatly appreciated by customers and colleagues.

As a freelance systems consultant for Borland Paradox, Matthias first came into contact with Borland products in the early 1990s and since 1997 worked as a permanent employee with many software products from Borland, Inprise, CodeGear, and Embarcadero. Of all the technologies, it was Delphi that he worked with since the very first release and that he had grown most fond of.

It is very poignant to us that Matthias passed away on February 14th, Valentine’s Day which is also the day we celebrate Delphi’s birthday, since his knowledge of Delphi had earned him the status of a legend in the Delphi community.

We wish Maria and all her relatives much strength in their grief.

From The Embarcadero Germany team with additional text from The Embarcadero US and International team.

We would also like to refer you to the following thread in Delphi Praxis

Delphi Praxis / https://www.delphipraxis.net/214652-eine-sehr-traurige-nachricht.html

Very sad news
Alt 15 Feb 2024, 12:47
It is with a mixture of sheer horror and great sadness that I have to inform you that our long-time community member Matthias Eißing passed away suddenly and completely unexpectedly on Wednesday night.

Many of you knew Matthias not only from this forum, of course, but will also have had professional dealings with Matthias, as he was a technical consultant with Embarcadero from the early Borland days until today. But of course we also knew him personally from countless congresses and live events as an excellent speaker. I myself had the honour of giving a few presentations with him, as a junior speaker so to speak, and it was a pleasure every single time. Matthias' technical enthusiasm, coupled with his seemingly inexhaustible wealth of Delphi knowledge, will be missed. There is so much I was able to learn from him - both professionally and personally. I can hardly list all the places where Matthias will leave a gigantic gap.

Matthias also introduced me to my current employer years ago, something I have never forgotten and will never forget.

Matthias, I am very, very sad that we will no longer have the opportunity to talk shop over a beer, drift off the subject and talk about something completely different afterwards and still have fun doing it. Have a good trip and rest in peace. It was wonderful to have got to know you.

In silent mourning,
Daniel
Daniel R. Wolf

Eine sehr traurige Nachricht
Alt 15. Feb 2024, 12:47
Mit einer Mischung aus blankem Entsetzen und großer Trauer muss ich Euch leider mitteilen, dass unser langjähriges Community-Mitglied Matthias Eißing in der Nacht zu Mittwoch plötzlich und völlig unerwartet verstorben ist.

Viele von Euch kannten Matthias natürlich nicht nur hier aus diesem Forum, sondern werden auch beruflich mit Matthias zutun gehabt haben, da er als technischer Berater seit den frühen Borland-Zeiten bis heute bei Embarcadero intensiven Kundenkontakt pflegte. Aber natürlich kannten wir ihn auch persönlich von unzähligen Kongressen und Live-Veranstaltungen als exzellenten Referenten. Ich selbst durfte einige Vorträge mit ihm halten, als Junior-Referent quasi, und es war mir jedes einzelne Mal eine Freude. Matthias technische Begeisterung, gepaart mit seinem unerschöpflich wirkenden Delphi-Wissensschatz wird fehlen. Es gibt so vieles, was ich von ihm lernen konnte - fachlich, aber auch menschlich. Ich vermag kaum aufzuzählen, wo überall Matthias einer gigantische Lücke hinterlassen wird.

Matthias brachte mich vor Jahren auch mit meinem heutigen Arbeitgeber zusammen, was ich ihm nie vergessen habe und auch nie vergessen werde.

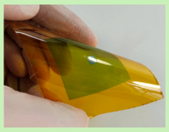
Matthias, ich bin sehr, sehr traurig, dass wir keine Gelegenheit mehr haben werden, bei einem Bier zu fachsimpeln, vom Thema abzudriften und hinterher über etwas ganz anderes zu sprechen und trotzdem Spaß dabei zu haben. Hab eine gute Reise und ruhe in Frieden. Es war wunderbar, Dich kennengelernt zu haben.

In stiller Trauer,
Daniel
Daniel R. Wolf

FLEXIBLE MICROPROCESSORS

BY DETLEF OVERBEEK

The original article was published By the MIT (Massachusetts Institute of Technology)



Silicon is the second most common element in the universe, after air. It makes up most of the computers we use today. Forms of silicon can be found in rocks, clay, sand, and dirt. Even though it's not the best, it's the easiest to get for electronics.
Because of this, silicon is the material that is most often found in electronics, like solar cells, sensors, and the chips inside computers and smartphones.
MIT engineers are working on a way to use a lot of different, non-silicon materials to make very **thin films that are also semiconducting.**

The researchers made bendable films of ***gallium arsenide, gallium nitride and lithium fluoride*** to show how their method worked. Even though these materials work better than silicon, they have been **too expensive** to use in functional products until now.

The experts say that the new method makes it easy and cheap to make flexible electronics from any mix of semiconducting parts. These circuits would work better than the ones we use now, which are based on silicon.

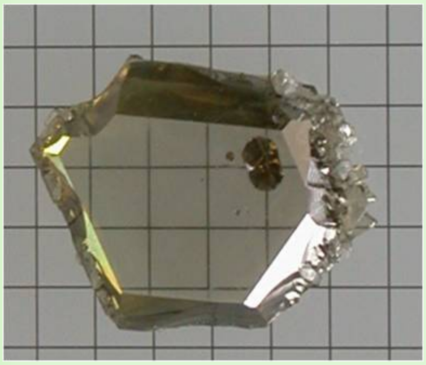


Figure 1: gallium nitride (mineral)

Jeehwan Kim, who works at **MIT** (Massachusetts Institute of Technology - Massachusetts Avenue / Cambridge MA) in both Mechanical Engineering and Materials Science and Engineering, stated,
"We have found a way to make flexible electronics that can be made from as many different types of materials as silicon." MIT supposes that this technology can be used to create low-cost gadgets that work well, like portable computers and sensors, as well as bendable solar cells.

HEXAGONAL CHICKEN WIRE

Kim and his coworkers came up with a way to "copy" expensive semiconductor materials with graphene in 2017.

WIKIPEDIA

*Graphene is an **allotrope*** of carbon consisting of a single layer of atoms arranged in a hexagonal lattice nanostructure. The name is derived from "graphite" and the suffix -ene, reflecting the fact that the graphite allotrope of carbon contains numerous double bonds.*

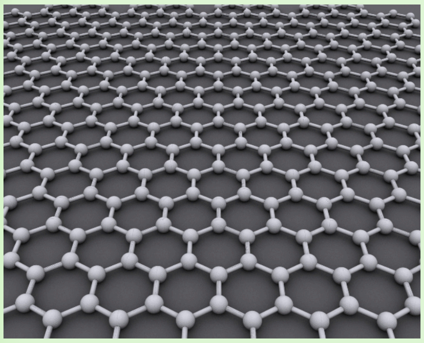
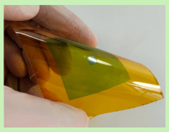


Figure 2: Hexagonal Chicken Wire of Graphene

allotropy, WIKIPEDIA
the existence of a chemical element in two or more forms, *which may differ in the arrangement of atoms in crystalline solids or in the occurrence of molecules that contain different numbers of atoms.*



Carbon is capable of forming many allotropes *(structurally different forms of the same element)* due to its **valency**. *Valency is the number of atoms of a particular element that is combined with one atom of another element to form a molecule. Valency is also known as molecular weight. Valency is a measure of the combining power of an atom. The valency of an element is determined by the number of electrons in its outermost shell.* Well-known forms of carbon include diamond and graphite. In recent decades, many more **allotropes** have been discovered and researched, including ball shapes such as **Buckminsterfullerene** *(It is a black solid that dissolves in hydrocarbon solvents to produce a violet solution.) and sheets such as graphene.*

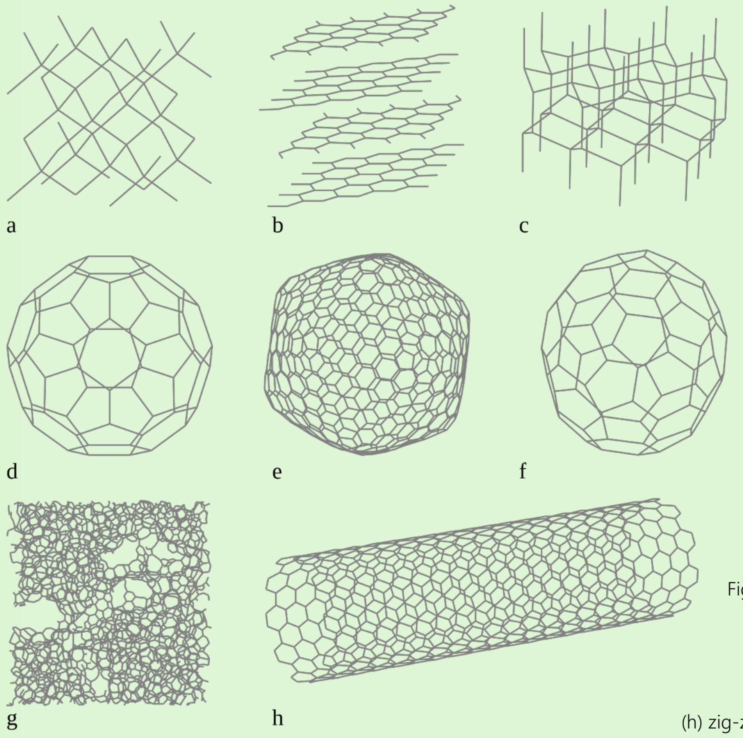


Figure 3: Eight **allotropes** of carbon:
(a) diamond,
(b) graphite,
(c) lonsdaleite,
(d) C60 buckminsterfullerene,
(e) C540 fullerite
(f) C70 fullerene,
(g) amorphous carbon,
(h) zig-zag single-walled carbon nanotube.

Graphene is made up of a very thin layer of carbon atoms grouped in a way that looks like *hexagonal chicken wire*.
Putting graphene on a pricey, pure wafer of a semiconducting material like gallium arsenide and then letting the atoms of gallium and arsenide flow across the wafer made it look like the atoms were interacting with each other in a way that made the graphene between them look like it wasn't there.
So, the atoms were able to stick together in the exact single-crystalline pattern of the semiconducting chip below. This let them make a perfect copy that was simple to peel off the graphene layer. The inexpensive method, which they called **"remote epitaxy (type of exposure)"**, made it possible to make many layers of gallium arsenide with just one pricey chip underneath.

The team thought if their method could be used to make other semiconducting materials soon after showing their first results. They tried remote epitaxy on silicon and germanium, which are both cheap semiconductors, but when they put the atoms on top of graphene, they didn't connect with the layers below. Suddenly, graphene went from being clear to opaque, making it impossible for silicon and germanium atoms to "see" the atoms on the other side.
Quartz and germanium are both in the same group of elements in the periodic table. To be more exact, these two elements are ionically neutral, which means they don't have any polarity. They are in the fourth class of materials. This showed to be a hint.

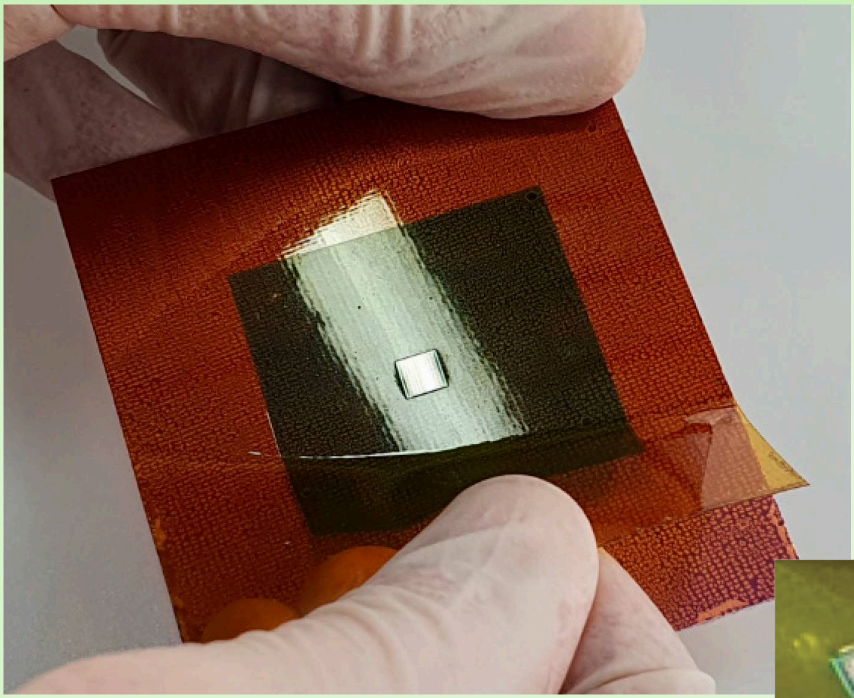


Figure 4: Peeling of the Microprocessor

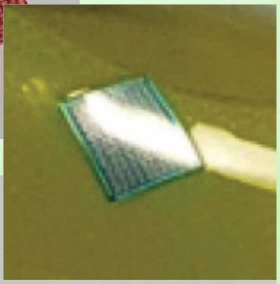


Figure 6: Enlarging the view

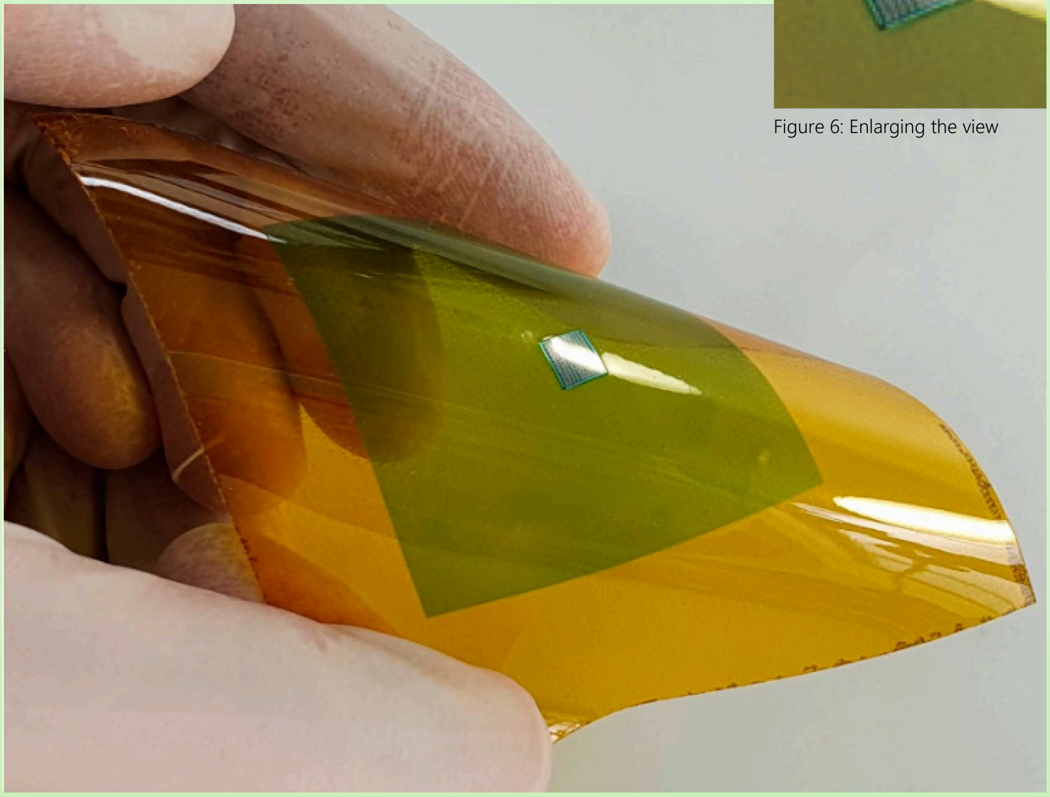
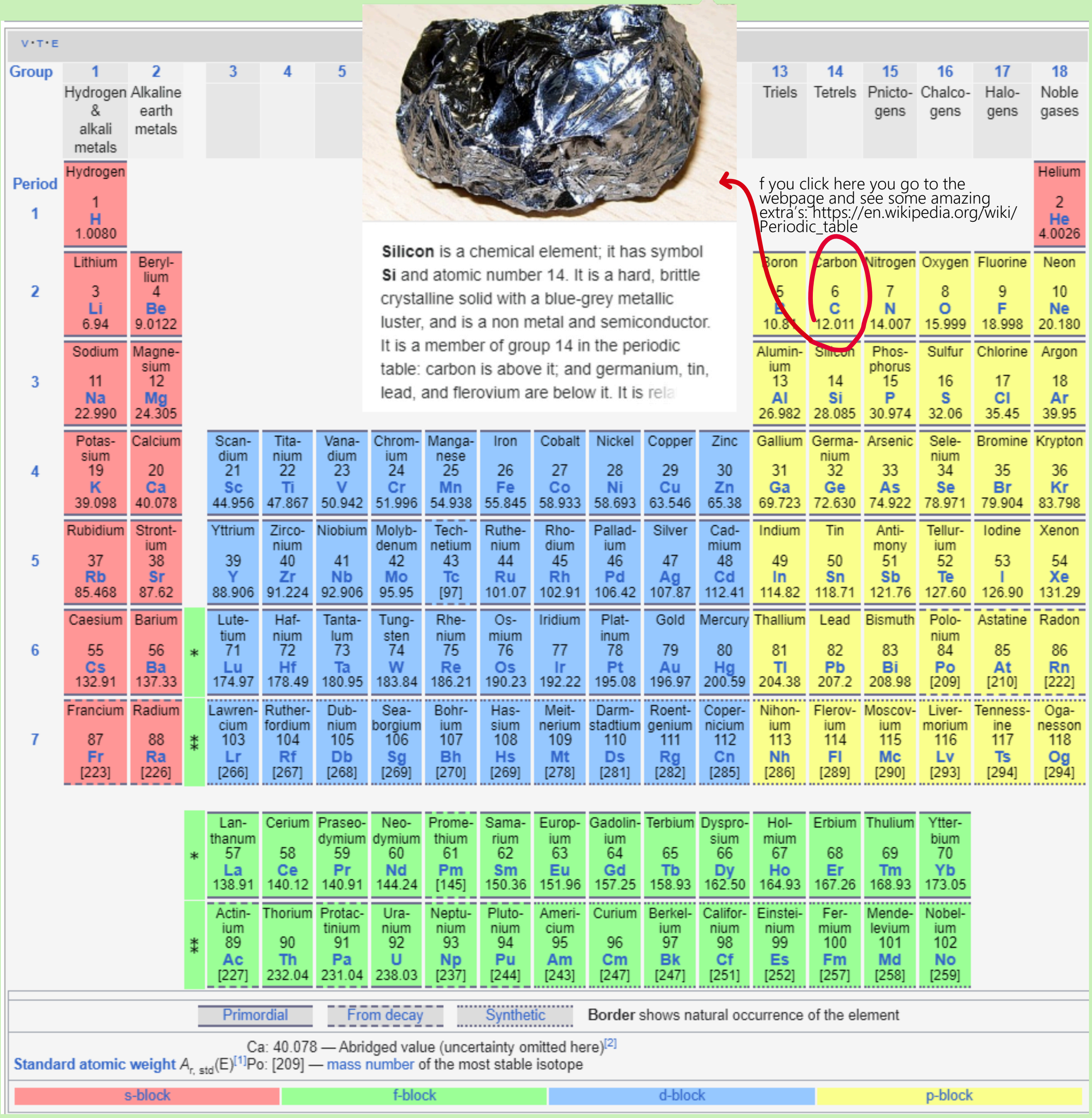


Figure 5: Notice the bending and flexibility



V·T·E

Group	1	2		3	4	5								13	14	15	16	17	18
	Hydrogen & alkali metals	Alkaline earth metals												Triels	Tetrels	Pictogens	Chalcogens	Halogens	Noble gases
Period 1	Hydrogen 1 H 1.0080																		Helium 2 He 4.0026
2	Lithium 3 Li 6.94	Beryllium 4 Be 9.0122												Boron 5 B 10.81	Carbon 6 C 12.011	Nitrogen 7 N 14.007	Oxygen 8 O 15.999	Fluorine 9 F 18.998	Neon 10 Ne 20.180
3	Sodium 11 Na 22.990	Magnesium 12 Mg 24.305												Aluminium 13 Al 26.982	Silicon 14 Si 28.085	Phosphorus 15 P 30.974	Sulfur 16 S 32.06	Chlorine 17 Cl 35.45	Argon 18 Ar 39.95
4	Potassium 19 K 39.098	Calcium 20 Ca 40.078		Scandium 21 Sc 44.956	Titanium 22 Ti 47.867	Vanadium 23 V 50.942	Chromium 24 Cr 51.996	Manganese 25 Mn 54.938	Iron 26 Fe 55.845	Cobalt 27 Co 58.933	Nickel 28 Ni 58.693	Copper 29 Cu 63.546	Zinc 30 Zn 65.38	Gallium 31 Ga 69.723	Germanium 32 Ge 72.630	Arsenic 33 As 74.922	Selenium 34 Se 78.971	Bromine 35 Br 79.904	Krypton 36 Kr 83.798
5	Rubidium 37 Rb 85.468	Strontium 38 Sr 87.62		Yttrium 39 Y 88.906	Zirconium 40 Zr 91.224	Niobium 41 Nb 92.906	Molybdenum 42 Mo 95.95	Technetium 43 Tc [97]	Ruthenium 44 Ru 101.07	Rhodium 45 Rh 102.91	Palladium 46 Pd 106.42	Silver 47 Ag 107.87	Cadmium 48 Cd 112.41	Indium 49 In 114.82	Tin 50 Sn 118.71	Antimony 51 Sb 121.76	Tellurium 52 Te 127.60	Iodine 53 I 126.90	Xenon 54 Xe 131.29
6	Caesium 55 Cs 132.91	Barium 56 Ba 137.33	*	Lutetium 71 Lu 174.97	Hafnium 72 Hf 178.49	Tantalum 73 Ta 180.95	Tungsten 74 W 183.84	Rhenium 75 Re 186.21	Osmium 76 Os 190.23	Iridium 77 Ir 192.22	Platinum 78 Pt 195.08	Gold 79 Au 196.97	Mercury 80 Hg 200.59	Thallium 81 Tl 204.38	Lead 82 Pb 207.2	Bismuth 83 Bi 208.98	Polonium 84 Po [209]	Astatine 85 At [210]	Radon 86 Rn [222]
7	Francium 87 Fr [223]	Radium 88 Ra [226]	**	Lawrencium 103 Lr [266]	Rutherfordium 104 Rf [267]	Dubnium 105 Db [268]	Seaborgium 106 Sg [269]	Bohrium 107 Bh [270]	Hassium 108 Hs [269]	Meitnerium 109 Mt [278]	Darmstadtium 110 Ds [281]	Roentgenium 111 Rg [282]	Copernicium 112 Cn [285]	Nihonium 113 Nh [286]	Flerovium 114 Fl [289]	Moscovium 115 Mc [290]	Livermorium 116 Lv [293]	Tennessine 117 Ts [294]	Oganesson 118 Og [294]
			*	Lanthanum 57 La 138.91	Cerium 58 Ce 140.12	Praseodymium 59 Pr 140.91	Neodymium 60 Nd 144.24	Promethium 61 Pm [145]	Samarium 62 Sm 150.36	Europium 63 Eu 151.96	Gadolinium 64 Gd 157.25	Terbium 65 Tb 158.93	Dysprosium 66 Dy 162.50	Holmium 67 Ho 164.93	Erbium 68 Er 167.26	Thulium 69 Tm 168.93	Ytterbium 70 Yb 173.05		
			**	Actinium 89 Ac [227]	Thorium 90 Th 232.04	Protactinium 91 Pa 231.04	Uranium 92 U 238.03	Neptunium 93 Np [237]	Plutonium 94 Pu [244]	Americium 95 Am [243]	Curium 96 Cm [247]	Berkelium 97 Bk [247]	Californium 98 Cf [251]	Einsteinium 99 Es [252]	Fermium 100 Fm [257]	Mendelevium 101 Md [258]	Nobelium 102 No [259]		

Silicon is a chemical element; it has symbol **Si** and atomic number 14. It is a hard, brittle crystalline solid with a blue-grey metallic luster, and is a non metal and semiconductor. It is a member of group 14 in the periodic table: carbon is above it; and germanium, tin, lead, and flerovium are below it. It is rela

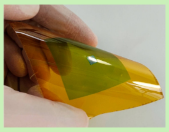
f you click here you go to the webpage and see some amazing extra's: https://en.wikipedia.org/wiki/Periodic_table

Primordial From decay Synthetic **Border** shows natural occurrence of the element

Standard atomic weight $A_{r,\,std}$(E)[1] Ca: 40.078 — Abridged value (uncertainty omitted here)[2] Po: [209] — mass number of the most stable isotope

s-block f-block d-block p-block

Figure 7: The periodic Table If you click here you can go to the webpage and see some amazing extra's



The group believed that graphene should have its own ionic charge so that it could combine with other molecules. Gallium arsenide, on the other hand, has a negative charge at the contact, while arsenic has a positive charge.
This difference in charge, or polarity, can help the atoms repeat the basic pattern of an atom and talk to each other through graphene as if it were clear.

What they discovered was that the way graphene interacts depends on how polar (*In chemistry, polarity is a separation of electric charge leading to a molecule or its chemical groups having an electric dipole moment, with a negatively charged end and a positively charged end.*) the atoms are. Even three layers of graphene can interact with each other for the strongest ionically bound materials. It works a lot like how two magnets can pull together through a thin piece of paper.

The team also discovered that it depends on the material in which the atoms interact.
They did experiments with both graphene and an inter-layer of **hexagonal boron nitride (hBN),** a substance that mimics the atomic pattern of graphene and has properties similar to those of **Teflon**, allowing overlying materials to peel off easily when copied.

But **HBN** is made of oppositely charged boron and nitrogen atoms, giving it polarity.
In their experiments, the researchers found that all the atoms that flew over **hBN**, even if they themselves were highly polarised, were unable to fully interact with their underlying wafers.

This suggested that both the polarity of the main atoms and the intermediate material determine whether the atoms will interact and form a copy of the original semiconducting wafer.

So it seems there are now rules for atomic interaction by graphene.
Researchers can now select only two elements with opposite charges by using this new understanding to look at the periodic table (*See Figure 7 on page 4 of this article*).
They can make multiple accurate copies of the original wafer by using remote epitaxy methods after making a main wafer from the same materials.

Silicon wafers are mostly used because they are cheap.
Using non-silicon materials is now made possible by this method.
You can simply buy one expensive wafer and copy it from time to time, allowing you to reuse the wafer. The material library for this method is now fully expanded.

It is predicted that **remote epitaxy** can now be used to produce ultra-thin, flexible films from a wide range of previously exotic semiconducting materials - as long as the materials are made of atoms with a certain polarity.
Even in the distant future, these ultra-thin films can be combined to make small, flexible, versatile devices such as **wearable sensors**, **flexible solar cells** and even "**mobile phones that stick to your skin**".
Kim says: "We need **low-energy, highly sensitive computing and sensor devices**, made from better materials, in smart cities, where we may want to put small computers everywhere." "This study opens the door to those devices."

POCKET PACKAGE (2BOOKS)   PRICE: € 40,00   EXCLUDING VAT AND SHIPPING

LAZARUS HANDBOOK

https://www.blaisepascalmagazine.eu/product-category/books/

PASCAL USERS TIPS SOLUTIONS

screen resolution in Windows

I found another way to get the screen resolution in Windows that is consistent between 32bit and 64bit programs using **GetSystemMetrics**.

```
uses
 Windows;

var
 ScreenWidth, ScreenHeight: Integer;

begin
 ScreenWidth  := GetSystemMetrics(SM_CXSCREEN);
 ScreenHeight := GetSystemMetrics(SM_CYSCREEN);

 WriteLn('Screen Width: ', ScreenWidth, ' pixels');
 WriteLn('Screen Height: ', ScreenHeight, ' pixels');
end.
```

This works well, and with PTCGraph, there is really no need to bother figuring out what video modes are available anyway, especially now that we can have custom resolutions for the PTCGraph window, I just need to make sure it fits on the screen so it really doesn't matter how I get the screen resolution. *James Richters*

How can I show the user that he has created an error?

This is often best done with an **Exception**. It allows you to step out of in an elegant manor out of your function or procedure.

It turns out to be very useful if you create two standard error procedures:

a. one that **beeps and jumps out** of the function/procedure

b. one that also puts an error text on the screen.

Example: Use a procedure with an **Abort-exception**:

```
PROCEDURE AbortWithBeep;
BEGIN
   MessageBeep(-1); Abort;
END;
```

Use a procedure with an Exception that puts an error message on the screen:

```
PROCEDURE ExceptOnMistake(Mistake : String);
BEGIN
   Messagebeep(-1);
RAISE Exception.Create(‘Error: ‘+ Mistake ;
END;
```

How can I ensure that the user can only enter numbers?

Let's assume for a moment that an Edit field is used.

Then you can check the input at two points:

a. or when the whole number is entered,

b. or as soon as a key is pressed.

For example, if you want to ensure that only (**real**) numbers are entered you create the following `OnExit` event:

```
PROCEDURE TForm1.Edit1Exit(Sender: TObject);
CONST LF=#13;{linefeed}
BEGIN
  IF Edit1.Text=” THEN
    BEGIN
      Edit1.SetFocus;
      ExceptOnError
        (’The amount is empty.’+LF+
        ’We need to get the amount’);
    END{if};

  TRY StrToFloat(Edit1.Text);
     EXCEPT
      Edit1.SetFocus;
      ExceptOnError
      (’The amount is not correctly typed.’+LF+
      ’Type only a number, minus or comma’);
     END{try};
END;
```

Edit1 is checked for errors at 'onExit'. The previously described `ExceptOnMistake` returns two error lines. The upper one names the error and the lower one gives the solution. This is a good practice.

The constant LF (line feed) is declared locally (within the procedure) declared.

In your program, it is better to declare it globally (at the beginning of your program), then you can use it anywhere.

If you want to ensure that integers are fed in, then use **StrToInt** instead of **StrToFloat**.

When it is plainly obvious which error has been made, a beep will suffice instead of an error text.

In that case, use the AbortWithBeep mentioned earlier in your procedure:

Edit1 is now checked on every keystroke with 'onKeyPress'. If an amount is to be typed in Edit1 then you can catch an accidentally typed letter with:

```
PROCEDURE TForm1.Edit2KeyPress
(Sender: TObject; VAR Key: Char);
BEGIN
   IF NOT (Key IN [#8,’,’,’-’,’0’..’9’])
   THEN  AbortWithBeep;
END;
```

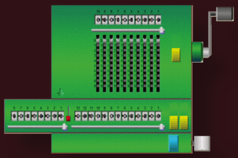
Key #8 ensures that the Backspace key continues to work.

You don't need to use the above procedures with a `DBEdit` field.

When the field is linked to a 'number' field in the Table, the entry is automatically validated.

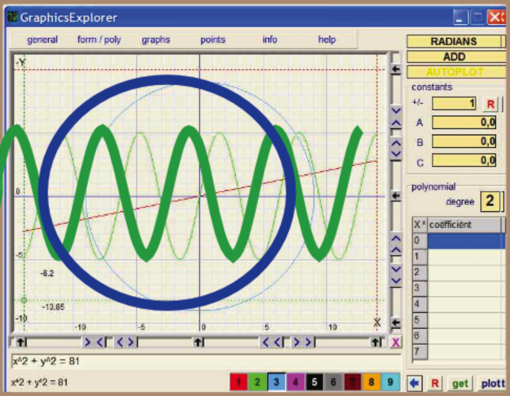
ADVERTISEMENT

David Dirkse's website: `davdata.nl/math`

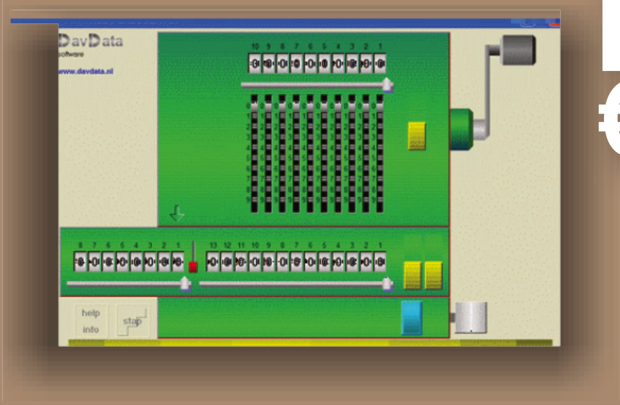


DAVID DIRKSE

including 50 example projects



x^2 + y^2 = 81



Pocket
€ 50
ex Shipping
PDF
€ 35

COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL

https://www.blaisepascalmagazine.eu/product-category/books/

NOT A NUMBER?

WHEN IS A NUMBER NOT A NUMBER?
WHEN IT IS A NaN.

BY DANNY WIND

DELPHI 12 NOW USES NaN AND INF

Starting with Delphi 12 a floating point calculation which would previously raise an exception will now result in a Not a Number (NaN) or INF value.

In **Delphi 12** all the **Floating Point Exceptions** are now masked by default. In previous **Delphi** versions three of the FPU Exceptions where unmasked, and a calculation might raise an **Invalid Operation**, **Zero Divide** or **Overflow** exception that needed to be handled in code with a try except. In **Delphi 12** for **VCL** these exceptions are no longer raised and we now get **NaN** and **INF** floating-point values as a result of invalid operation and division by zero and overflow calculations.

FPU detection	Calculation	<= Delphi 11.3 (VCL)	>= Delphi 12 (VCL)
exInvalidOp	0.0 / 0.0 sqrt(-1)	EInvalidOp	NaN
exDenormalized	0.01 / 1e308	9,99999999999997E-311	9,99999999999997E-311
exZeroDivide	1.0 / 0.0	EZeroDivide	INF
exOverflow	1e308 + 1e308	EOverflow	INF
exUnderflow	1e-16 / 1e308	0.0	0.0
exPrecision	Most calculations	Closest round ties to even value	Closest round ties to even value

Note that this mostly affects **VCL** applications, as **FMX** applications already used the new exception model. It does also affect **FMX** a bit though, as underlying code now also more closely follows **IEEE 754**.

WHAT'S NOT A NUMBER

The concept of NaN, or Not a Number in response to 0.0/0.0 or sqrt(-1) may be surprising at first, but it soon starts to make sense after some careful thought.

When **William Kahan, the Father of Floating Point**, wrote the **IEEE 754** rulebook for floatingpoint numbers he did so with care and insight. One of these insights was including the systematic use of a NaN value, where an invalid result of part of a calculation such as 0.0/0.0 would be allowed to yield a special floating-point value, a NaN, instead of halting the calculation with an exception.
In this way all the following and surrounding calculations could continue and the **NaN** would become part of this calculation. The NaN would propagate to parts of the end-result, following specific rules. In this way calculations could still yield perfectly valid partial results, for instance with matrices, even if some of the steps had NaN values. This works the same with INFinity values.

In the **IEEE 754** specification it is written that you can also handle these and other boundary conditions, with exceptions. Back then **Delphi** and several other programming languages chose to do just that. But gradually over the years, more and more of the programming languages adopted the use of NaN and INF values over exceptions.
Why?
Well, it leads to faster calculations and allows calculations, especially matrix multiplications, to fail just partially, still yielding correct results outside of the boundary conditions. As now most programming languages have moved over to this approach it is now time for Delphi to do the same.

With the new **Delphi 12** we fully embrace NaN and INF and do away with exceptions while running calculations.
However because of that we also have to know more about these special values. And maybe its also a good thing to freshen up on our general floating point knowledge.

NAN SPECIAL CONSIDERATIONS

If you look at a NaN it is essentially an unknown number.

This means that any comparison with another number or even with a NaN would make no sense. It could be both True and False. In IEEE 754 the result of a comparison with NaN is suggested to always result in a False result. Note that this behaviour is not mandatory, each and every language and CPU platform may choose to implement NaN comparisons differently.

In calculations, anything that you calculate with a NaN should result in a NaN as well.
Or as put forward in IEE 754: "For an operation with quiet NaN inputs, other than maximum and minimum operations, if a floating-point result is to be delivered the result shall be a quiet NaN which should be one of the input NaNs. "

Comparison between NaN and other numbers

NaN <> x	True	unordered	NOT(NaN <> x)	False
NaN = x	False	means	NOT(NaN = x)	True
NaN > x	False	it	NOT(NaN > x)	True
NaN >= x	False	has	NOT(NaN >= x)	True
NaN < x	False	no	NOT(NaN < x)	True
NaN <= x	False	order	NOT(NaN <= x)	True

The interpretation of NaN comparisons is unordered, as you can not really know what number NaN would really represent. The only comparison that would be true is that when x is a normal number it is not equal to a NaN. All other comparisons are False.

NAN UNEXPECTED RESULTS

This unordered definition leads to (NaN > x) being evaluated as False while NOT(NaN <=x) evaluates to True. In software this could easily lead to unexpected results.
For instance in cruise control software for a vehicle. If the code uses something like this;

```
while cruise control enabled do
begin
  if NOT(MeasuredSpeed >= TargetSpeed) then
     {keep adding 0.1 to acceleration to get up to speed}
  else
     {reduce acceleration to -0.1 and stabilize}
end;
```

And if the designer of the the speed-o-meter decided to return NaN if the measurement is invalid, you would soon be going faster than you intended to.

DEBUGGING NAN AND INF ISSUES

It is perfectly valid to re-enable the good-old FPU exceptions for debugging purposes.
This can easily be done by using the following code

```
uses System.Math;
    SetExceptionMask(exAllArithmeticExceptions-LegacyExceptionFlags);
```

After re-enabling these exceptions you can throw all kinds of unit, regression and other tests you can come up with at your software to weed out all these exceptions.
At this point you might be wondering if it would not be easier to leave all the old code as is and just re-enable these legacy exceptions. Its not a bad thought, however it does come with some pitfalls. One of them is that the new **Delphi 12** base code more closely follows IEE 754 in general and you can not entirely go back to the old behaviour.
For instance in comparisons with NaN **Delphi 11.3** and **Delphi 12** behave differently.

NaN > x	Platform	<= Delphi 11.3	>= Delphi 12
	VCL – Win32	True	False
	VCL – Win64	EInvalidOp	False
	FMX – Win32	True	False
	FMX – Win64	False	False

The results you get with **Delphi 12** are arguably better and conform to **IEEE 754**.
You can verify this yourself with **Delphi 11.3** and **Delphi 12** with the following code snippet.

```
function NaN_larger_x: Boolean;
var
  lNaN, x: Double;
begin
  lNan := Double.NaN;
  x := 42.0;
  if (lNaN > x) then
    Result := True
  else
    Result := False;
end;
```

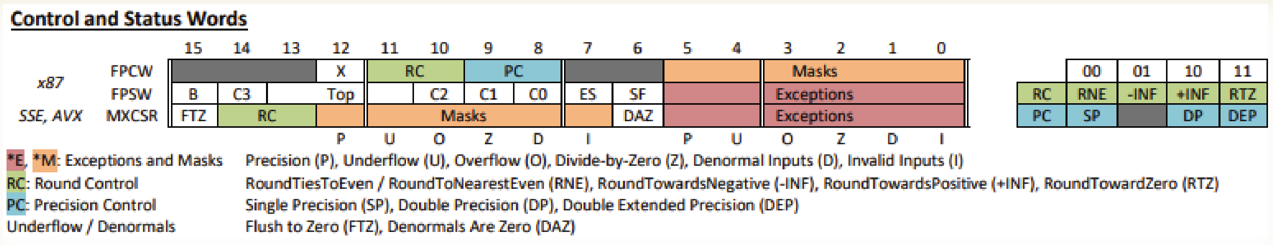
FPU CONTROL WORD

The default for the **FPU 8087** Control Word has also changed in **Delphi 12**. This value only has meaning for **Win32 (x86 32-bits)** and influences rounding, precision and exceptions.

FPU Control Word	<= Delphi 11.3	>= Delphi 12
VCL – Win32	1372	Default 037F
	0001 0011 0111 0010	0000 0011 0111 1111
		Legacy 0372
		0000 0011 0111 0010
FMX – Win32	137F	Default 037F
	0001 0011 0111 1111	0000 0011 0111 1111
		Legacy 0372
		0000 0011 0111 0010

The **Delphi 12** column lists both the **8087 CW** value when all **FPU** exceptions are masked (Default) and when the three legacy **FPU** exceptions are unmasked (Legacy).
We expect some of these changes, as the (un)masking of exceptions is part of the **8087 Control Word.**
The exception mask consists of the last 6 bits on the right of the **8087 CW** and these values are as expected.
With all exceptions masked these bits are all 1, for the legacy setting three of them are zero.

But what is happening with the fourth bit from the left?
Why is that bit now a zero instead of a one?



Control and Status Words

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x87	FPCW				X	RC		PC						Masks			
	FPSW	B	C3		Top		C2	C1	C0	ES	SF			Exceptions			
SSE, AVX	MXCSR	FTZ	RC			Masks					DAZ			Exceptions			
					P	U	O	Z	D	I		P	U	O	Z	D	I

	00	01	10	11
RC	RNE	-INF	+INF	RTZ
PC	SP		DP	DEP

*E, *M: Exceptions and Masks	Precision (P), Underflow (U), Overflow (O), Divide-by-Zero (Z), Denormal Inputs (D), Invalid Inputs (I)
RC: Round Control	RoundTiesToEven / RoundToNearestEven (RNE), RoundTowardsNegative (-INF), RoundTowardsPositive (+INF), RoundTowardZero (RTZ)
PC: Precision Control	Single Precision (SP), Double Precision (DP), Double Extended Precision (DEP)
Underflow / Denormals	Flush to Zero (FTZ), Denormals Are Zero (DAZ)

From the floating-point reference sheet for intel architecture we learn that bit 12 is a reserved bit, X marks the spot. With a bit of digging in the past it would seem that on 80387 CPUs settings this bit was reserved and used to enable **positive and negative infinity** instead of **just single infinity**.
Other sources suggest it can be used to set **denormals** to be flushed to zero or saved.
However the 8087 seems to have never supported flushing denormals to zero,
it is supported on **SSE** though.
Based on this info, the current spec and some tests I suspect this bit does nothing anymore, and this is just a bit of cleanup.

As a side note, the value **`037F`** is the default value that Intel uses for the **8087 CW**.

I'm going to stop here, although there is much more to know and learn about **floatingpoint** numbers. It still happens that floating-point calculation surprises me and it may also surprise you from time to time.
I'm going to sign off with this one fun fact:

FUN FACT

Did you know that if you keep adding 1.0 to a Double value you eventually end up with 9.007.199.254.740.992 ?
You may think that this is the maximum value for a Double, but it is not.
It's just the value where the precision of the Double becomes less than 2,
and the succeeding value+1 is rounded back to even to the nearest and exact same value.
You can go higher, but you have to add more than just 1 to get to the next floating-point value.
The step or interval between floating-point values is known as the ULP, the Unit in the Last Place.
This also means that after this number there are only even numbers.

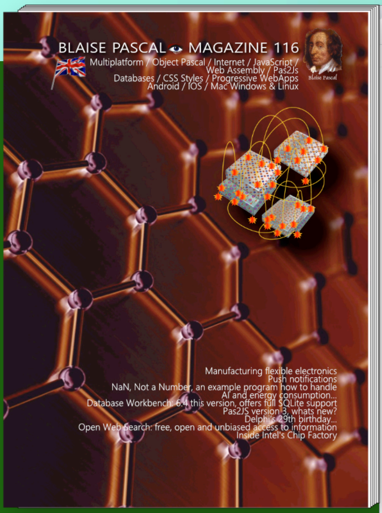
USEFUL LINKS AND REFERENCES (just click on it)

`https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html`

`https://www.intel.com/content/www/us/en/developer/articles/technical/floating-pointreference-sheet-for-intel-architecture.html`

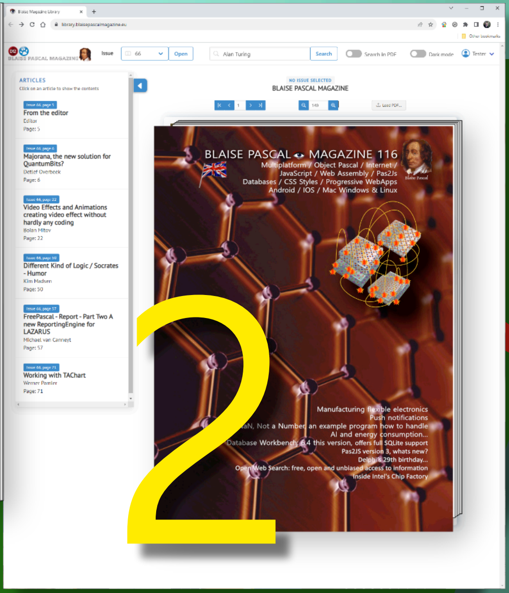
ADVERTISEMENT

1


BLAISE PASCAL MAGAZINE 116
Multiplatform / Object Pascal / Internet / JavaScript / Web Assembly / Pas2JS
Databases / CSS Styles / Progressive WebApps
Android / IOS / Mac Windows & Linux

Manufacturing flexible electronics
Push notifications
NaN, Not a Number, an example program how to handle
AI and energy consumption...
Database Workbench 6.4 this version, offers full SQLite support
Pas2JS version 3, whats new?
Delphi's 29th birthday...
Open Web Search: free, open and unbiased access to information
Inside Intel's Chip Factory

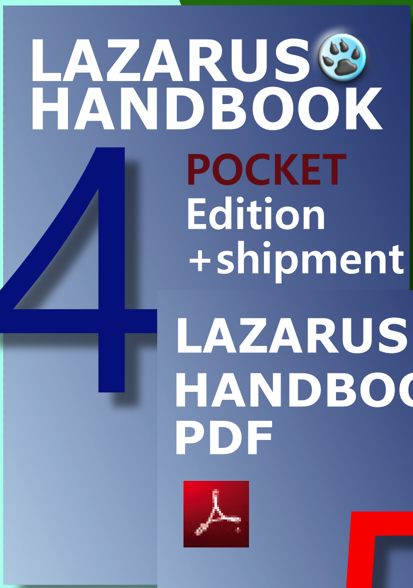
2



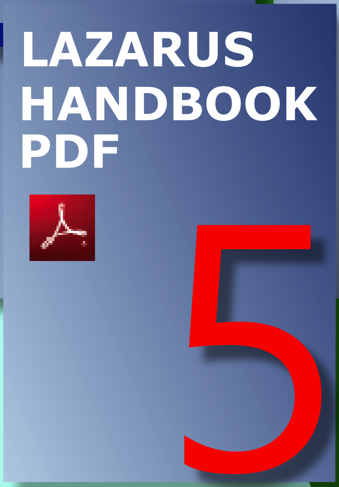
ARTICLES
Click on an article to show the contents
From the editor
Majorana, the new solution for QuantumBits?
Video Effects and Animations creating video effect without hardly any coding
Different Kind of Logic / Socrates - Humor
FreePascal - Report - Part Two A new ReportingEngine for LAZARUS
Working with TAChart
NO ISSUE SELECTED
BLAISE PASCAL MAGAZINE

3

4


LAZARUS HANDBOOK
POCKET Edition +shipment

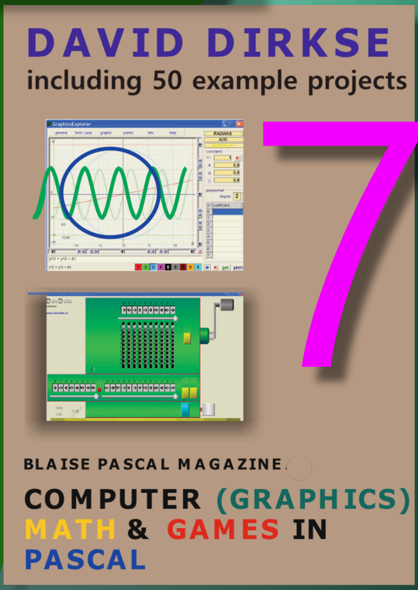
5

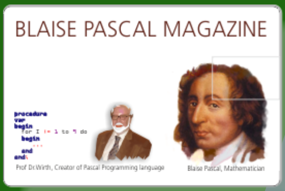

LAZARUS HANDBOOK PDF

6

LEARN TO PROGRAM USING LAZARUS
HOWARD PAGE-CLARK

7


DAVID DIRKSE
including 50 example projects
BLAISE PASCAL MAGAZINE
COMPUTER (GRAPHICS) MATH & GAMES IN PASCAL


BLAISE PASCAL MAGAZINE

Prof Dr.Wirth, Creator of Pascal Programming language
Blaise Pascal, Mathematician

BLAISE PASCAL MAGAZINE

Prof Dr.Wirth, Creator of Pascal Programming language
Blaise Pascal, Mathematician

Editor in Chief: Detlef Overbeek
Edelstenenbaan 21 3402 XA
IJsselstein Netherlands
Prof Dr.Wirth, Creator of Pascal Programming language
editor@blaisepascalmagazine.eu
https://www.blaisepascalmagazine.eu

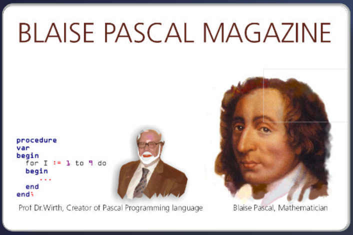
1. One year Subscription
2. Internet Viewing of the Magazine
3. The newest LIB Stick
   - All issues 1-111
   - On Credit Card
4. Lazarus Handbook Pocket
5. LH PDF including Code
6. Book Learn To Program
   - using Lazarus PDF including 19 lessons and projects
7. Book Computer Graphics Math & Games
   - PDF including ±50 projects

SPRING OFFER
SUPER PACK
7 ITEMS

PRICE € 120
NORMAL PRICE € 275

LIB-STICK ON USB CREDIT CARD BLAISE PASCAL MAGAZINE

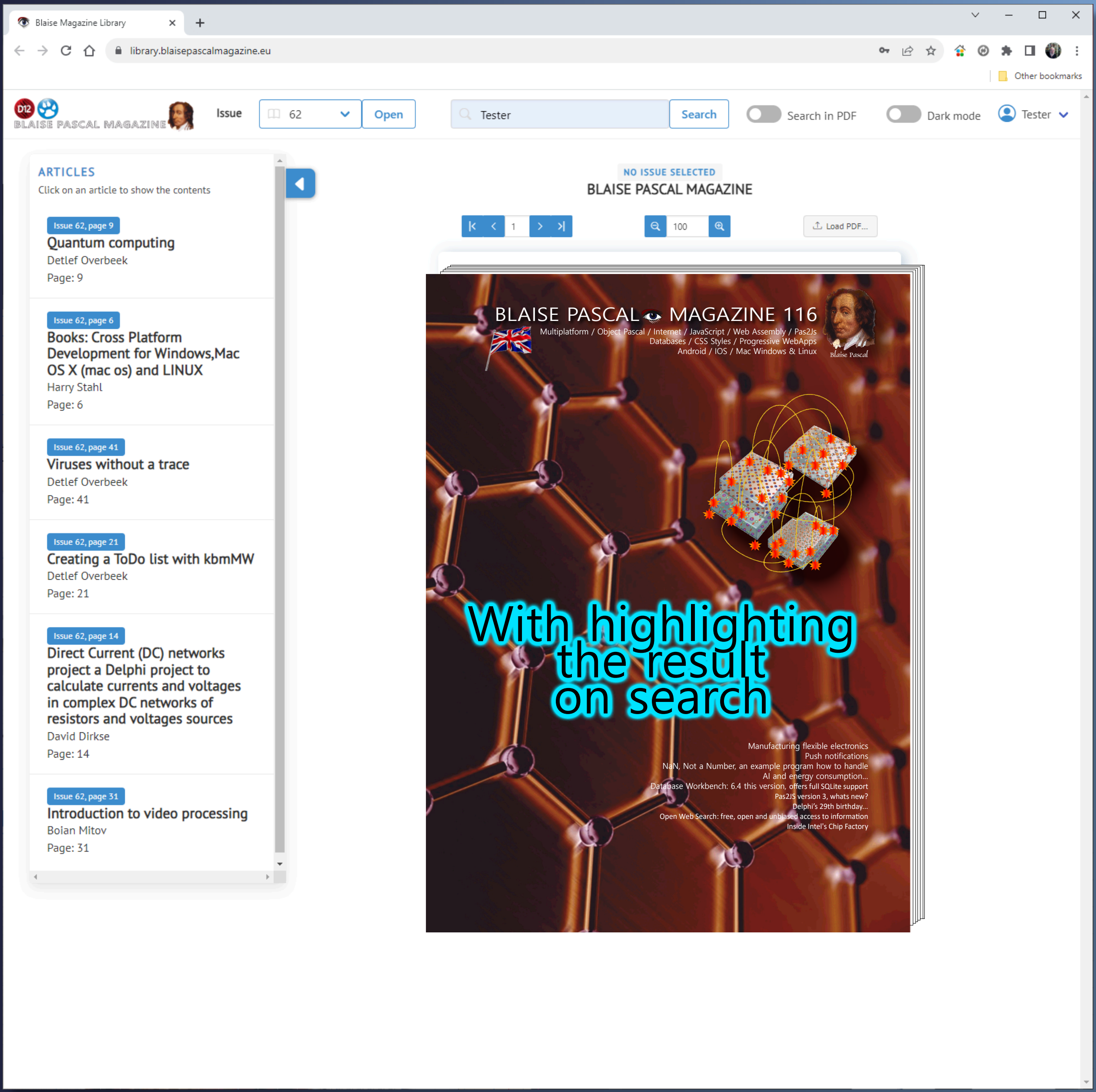
LIB-STICK USB-CARD: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY € 100



BLAISE PASCAL MAGAZINE
procedure
var
begin
for I := 1 to 9 do
begin
...
end
end;
Prof Dr.Wirth, Creator of Pascal Programming language
Blaise Pascal, Mathematician

Editor in Chief: Detlef Overbeek
Edelstenenbaan 21 3402 XA
IJsselstein Netherlands
Prof Dr.Wirth, Creator of Pascal Programming language
editor@blaisepascalmagazine.eu
https://www.blaisepascalmagazine.eu

BLAISE PASCAL MAGAZINE
procedure
var
begin
for I := 1 to 9 do
begin
...
end
end;
Prof Dr.Wirth, Creator of Pascal Programming language
Blaise Pascal, Mathematician



ARTICLES

Click on an article to show the contents

Issue 62, page 9
Quantum computing
Detlef Overbeek
Page: 9

Issue 62, page 6
Books: Cross Platform Development for Windows,Mac OS X (mac os) and LINUX
Harry Stahl
Page: 6

Issue 62, page 41
Viruses without a trace
Detlef Overbeek
Page: 41

Issue 62, page 21
Creating a ToDo list with kbmMW
Detlef Overbeek
Page: 21

Issue 62, page 14
Direct Current (DC) networks project a Delphi project to calculate currents and voltages in complex DC networks of resistors and voltages sources
David Dirkse
Page: 14

Issue 62, page 31
Introduction to video processing
Boian Mitov
Page: 31

NO ISSUE SELECTED

BLAISE PASCAL MAGAZINE

BLAISE PASCAL MAGAZINE 116
Multiplatform / Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js
Databases / CSS Styles / Progressive WebApps
Android / IOS / Mac Windows & Linux
Blaise Pascal

With highlighting the result on search

Manufacturing flexible electronics
Push notifications
NaN, Not a Number, an example program how to handle
AI and energy consumption...
Database Workbench: 6.4 this version, offers full SQLite support
Pas2JS version 3, whats new?
Delphi's 29th birthday...
Open Web Search: free, open and unbiased access to information
Inside Intel's Chip Factory

WHAT’S THE CHALLENGE?

A BILLION IS COOL

BY IAN BARKER, EMBARCADERO DEVELOPER ADVOCATE

There is a fantastic scene in The Social Network movie during Eduardo Saverin's pre-trial deposition where he recalls a moment when he and Marc Zuckerberg meet Sean Parker, the founder of Napster for the first time. Zuck, Eduardo, and Sean are discussing monetization of "The Facebook" as it was still called at the time. In the fictional scene, after Marc and Eduardo suggest that the site could be worth a substantial amount of money Sean Parker delivers an immortal line: "Yeah sure, you could do all that and get a valuation of a million dollars. But you know what's really cool? A BILLION dollars".

Why a million is not enough
That's the crux of things. Nowadays our societal sophistication is such that the word "million" is almost passè. A million dollar valuation of a US business would be considered almost tiny. Sending a spaceship on a million mile journey is just not that incredible any more. Been there, done that. This goes for the world of computing too where having a database with a million records is cool but, actually, there are data breaches which are regularly in the tens of millions.

To paraphrase Sean Parker, do you know what is cool? A database with a BILLION records.

THE ATTRACTION OF THE DARK SIDE

Normally I'm not really that lit up by coding challenges. To me they have a tendency to be a form of intellectual onanism. They can often be elitist and have a kind of "you're not in our gang" frat-house vibe precluding participation. Since my raison d'etre is about getting as many new people as possible to use Delphi I lean towards things which get new people to see what Object Pascal can do for them and show them why I think it's the best choice in almost any situation. Actions speak louder than words. I'm not going to be able to lure unsuspecting new coders to the florid charms of being/end statements and strong types by fussing over esoteric exchanges about CPU cycles and maven geek speak about re-entrant black magic frippery am I?

But on the other side of this, Object Pascal, the language we all love, is also alive, capable, and can kick many of its peers in their soft and squishy bits when it comes to speed, code safety, and a sublime succinctness of expression. So, with that in mind, are you in the mood to flex your coding muscles in anger and really wield Object Pascal like the precision-guided weapon of choice it is? You are? Then welcome to the Billion Row Challenge.

WHAT IS THE BILLION ROW CHALLENGE?

This came about thanks to Embarcadero MVP Gus Carreno. Here's how he describes things:

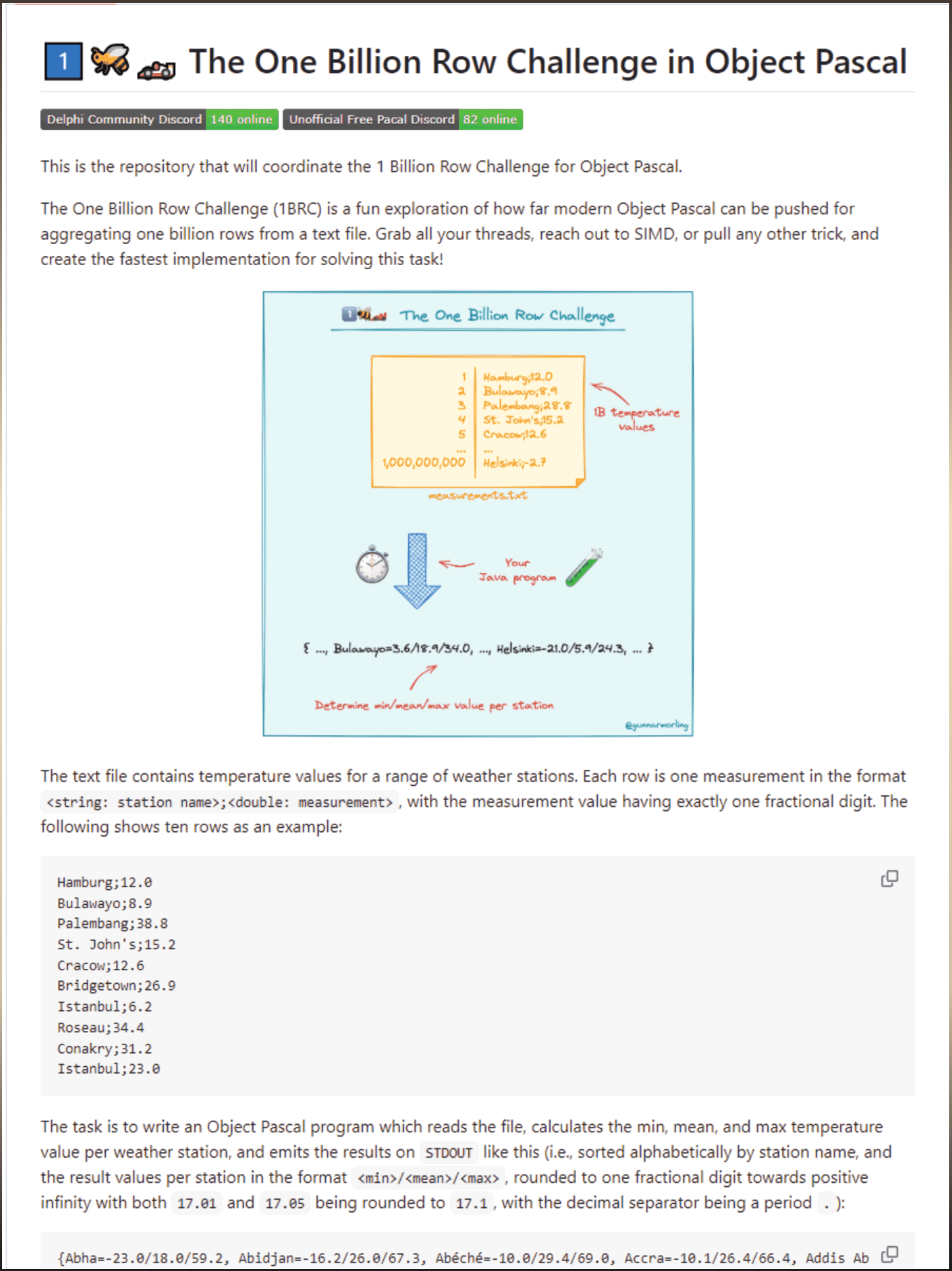
I've stumbled on a very interesting challenge: Reading 1 billion rows of text, processing some temperature values, ordering the collated values and then outputting them.
Do it single thread, do it multithread, but do it as fast as possible!

The original idea came from the Java world: 1BRC in Java
`https://github.com/gunnarmorling/1brc`

I immediately thought this would be a wonderful, and fun, challenge to do in Object Pascal.

If you think that this is a thing you'd enjoy, please head on to the 1BRC in Object Pascal repo which can be found here:
`https://github.com/gcarreno/1brc-ObjectPascal`

The fundamentals of this challenge are described there, but I can summarize it in a single sentence: Learn something, teach something, but please have fun doing so.

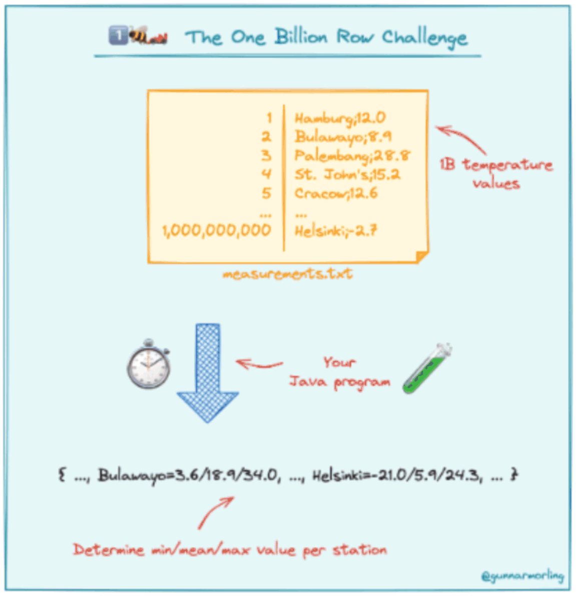


1 🐝 🏎️ The One Billion Row Challenge in Object Pascal

Delphi Community Discord 140 online   Unofficial Free Pacal Discord 82 online

This is the repository that will coordinate the 1 Billion Row Challenge for Object Pascal.

The One Billion Row Challenge (1BRC) is a fun exploration of how far modern Object Pascal can be pushed for aggregating one billion rows from a text file. Grab all your threads, reach out to SIMD, or pull any other trick, and create the fastest implementation for solving this task!

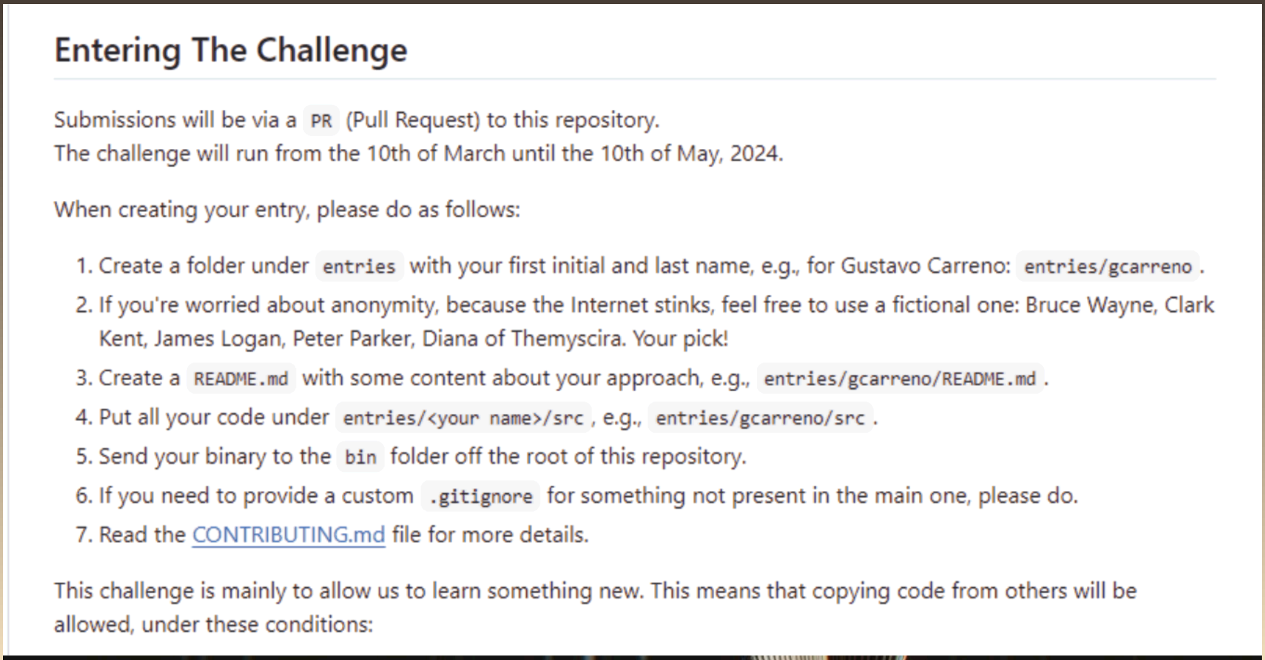

The One Billion Row Challenge
1 Hamburg;12.0
2 Bulawayo;8.9
3 Palembang;28.8
4 St. John's;15.2
5 Cracow;12.6
...
1,000,000,000 Helsinki;-2.7
1B temperature values
measurements.txt
Your Java program
{ ..., Bulawayo=3.6/18.9/34.0, ..., Helsinki=-21.0/5.9/24.3, ... }
Determine min/mean/max value per station
@gunnarmorling

The text file contains temperature values for a range of weather stations. Each row is one measurement in the format `<string: station name>;<double: measurement>`, with the measurement value having exactly one fractional digit. The following shows ten rows as an example:

```
Hamburg;12.0
Bulawayo;8.9
Palembang;38.8
St. John's;15.2
Cracow;12.6
Bridgetown;26.9
Istanbul;6.2
Roseau;34.4
Conakry;31.2
Istanbul;23.0
```

The task is to write an Object Pascal program which reads the file, calculates the min, mean, and max temperature value per weather station, and emits the results on `STDOUT` like this (i.e., sorted alphabetically by station name, and the result values per station in the format `<min>/<mean>/<max>`, rounded to one fractional digit towards positive infinity with both `17.01` and `17.05` being rounded to `17.1`, with the decimal separator being a period `.`):

```
{Abha=-23.0/18.0/59.2, Abidjan=-16.2/26.0/67.3, Abéché=-10.0/29.4/69.0, Accra=-10.1/26.4/66.4, Addis Ab
```



Entering The Challenge

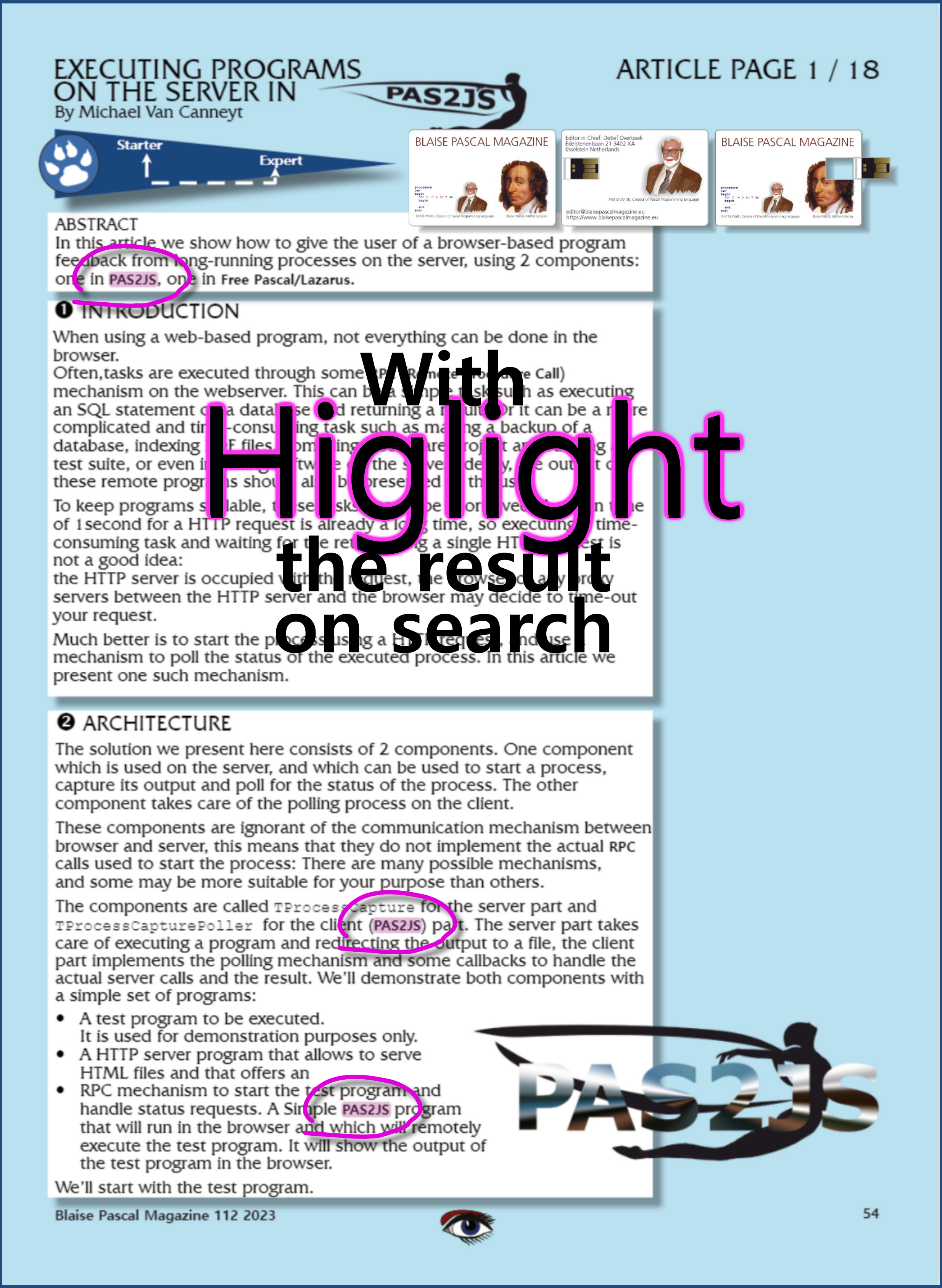
Submissions will be via a `PR` (Pull Request) to this repository.
The challenge will run from the 10th of March until the 10th of May, 2024.

When creating your entry, please do as follows:

1. Create a folder under `entries` with your first initial and last name, e.g., for Gustavo Carreno: `entries/gcarreno`.
2. If you're worried about anonymity, because the Internet stinks, feel free to use a fictional one: Bruce Wayne, Clark Kent, James Logan, Peter Parker, Diana of Themyscira. Your pick!
3. Create a `README.md` with some content about your approach, e.g., `entries/gcarreno/README.md`.
4. Put all your code under `entries/<your name>/src`, e.g., `entries/gcarreno/src`.
5. Send your binary to the `bin` folder off the root of this repository.
6. If you need to provide a custom `.gitignore` for something not present in the main one, please do.
7. Read the CONTRIBUTING.md file for more details.

This challenge is mainly to allow us to learn something new. This means that copying code from others will be allowed, under these conditions:

LIB-STICK BLAISE PASCAL MAGAZINE



EXECUTING PROGRAMS ON THE SERVER IN PAS2JS
By Michael Van Canneyt

ARTICLE PAGE 1 / 18

Starter Expert

ABSTRACT
In this article we show how to give the user of a browser-based program feedback from long-running processes on the server, using 2 components: one in PAS2JS, one in Free Pascal/Lazarus.

❶ INTRODUCTION
When using a web-based program, not everything can be done in the browser.
Often,tasks are executed through some RPC (Remote Procedure Call) mechanism on the webserver. This can b[illegible] a s[illegible]mp[illegible] task such as executing an SQL statement o[illegible] a data[illegible]se [illegible]d returning a r[illegible]ult. Or it can be a m[illegible]re complicated and ti[illegible]-consu[illegible]ing task such as ma[illegible]ng a backup of a database, indexing [illegible]F files, [illegible]om[illegible]ing [illegible]are [illegible]roj[illegible]t a[illegible] [illegible]g test suite, or even i[illegible] [illegible]ftw[illegible] [illegible] the [illegible]ve[illegible] de[illegible]y, [illegible]e out[illegible]t o[illegible] these remote progr[illegible]s shou[illegible] al[illegible] b[illegible] prese[illegible]ed [illegible] th[illegible] us[illegible]
To keep programs s[illegible]lable, t[illegible]se[illegible] sks [illegible] be [illegible] or[illegible] ve[illegible] n the of 1second for a HTTP request is already a lo[illegible]g time, so executing [illegible] time-consuming task and waiting f[illegible] the re[illegible]g a single HT[illegible]est is not a good idea:
the HTTP server is occupied [illegible] th[illegible] request, t[illegible]e browse[illegible] or a[illegible]y proxy servers between the HTTP server and the browser may decide to time-out your request.
Much better is to start the p[illegible]ocess using a [illegible] request and use a mechanism to poll the status of the executed process. In this article we present one such mechanism.

With
Higlight
the result
on search

❷ ARCHITECTURE
The solution we present here consists of 2 components. One component which is used on the server, and which can be used to start a process, capture its output and poll for the status of the process. The other component takes care of the polling process on the client.
These components are ignorant of the communication mechanism between browser and server, this means that they do not implement the actual RPC calls used to start the process: There are many possible mechanisms, and some may be more suitable for your purpose than others.
The components are called `TProcessCapture` for the server part and `TProcessCapturePoller` for the client (PAS2JS) part. The server part takes care of executing a program and redirecting the output to a file, the client part implements the polling mechanism and some callbacks to handle the actual server calls and the result. We'll demonstrate both components with a simple set of programs:
- A test program to be executed. It is used for demonstration purposes only.
- A HTTP server program that allows to serve HTML files and that offers an
- RPC mechanism to start the test program and handle status requests. A Simple PAS2JS program that will run in the browser and which will remotely execute the test program. It will show the output of the test program in the browser.

We'll start with the test program.

Blaise Pascal Magazine 112 2023 54

BLAISE PASCAL MAGAZINE
Editor in Chief: Detlef Overbeek
Edelstenenbaan 21 3402 XA
IJsselstein Netherlands
editor@blaisepascalmagazine.eu
https://www.blaisepascalmagazine.eu
BLAISE PASCAL MAGAZINE

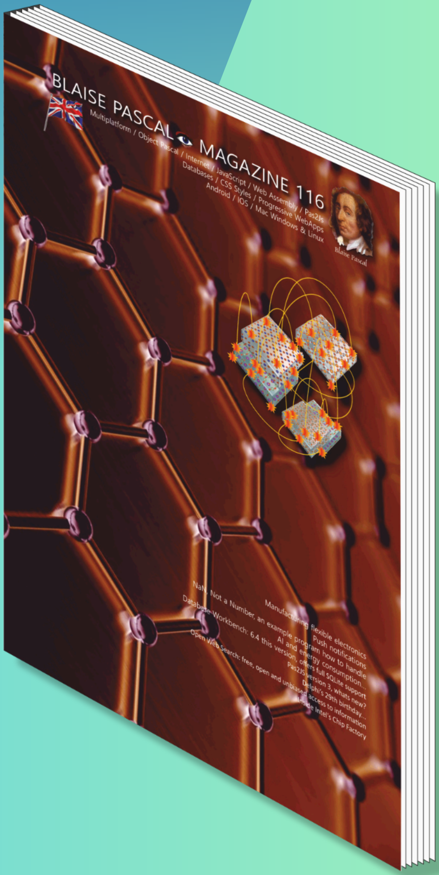
LIB-STICK USB-CARD: ALL ISSUES / CODE INCLUDED. SAME INTERFACE AS THE INTERNET LIBRARY € 89 CHRISTMAS OFFER

LAZARUS HANDBOOK (PDF) +SUBSCRIPTION 1 YEAR

- **Lazarus Handbook**
- Printed in black and white
- PDF Index for keywords
- Almost 1000 Pages
- Including 40 Examples
- **Blaise Pascal Magazine**
- English and German
- Free Lazarus PDF Kit Indexer
- 8 Issues per year
- minimal 60 pages
- Including example projects and code

SPECIAL OFFER € 75

+


BLAISE PASCAL MAGAZINE 116
Multiplatform / Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js
Databases / CSS Styles / Progressive WebApps
Android / IOS / Mac Windows & Linux

EXTENDED RTTI SUPPORT IN FREE PASCAL

BY MICHAEL VAN CANNEYT

ABSTRACT

For a long time, **Free Pascal** lacked support for **Extended RTTI.** Recently, the **Extended RTTI** support has been merged to the main development tree, making **Free Pascal** again more compatible to **Delphi**. In this article we take a closer look.

❶ INTRODUCTION

Introspection is the ability to examine the structure of data (*or types*) at runtime, without knowing exactly what type the data is:
it allows you to examine the type using some API provided by the programming language. In **Delphi** and **Free Pascal**, the introspection mechanism is labeled **RTTI**:
Run-Time Type Information.

Since its inception, Delphi has supported a limited form of RTTI:
The published properties of a class could (*and still can*) be examined with the aid of the **RTTI API:** at first the **`TypInfo`** unit, later the **`System.Rtti`** unit. This mechanism is used to implement streaming, and streaming is what makes form files work, both for **FMX** and the **VCL**.

Free Pascal has supported this form of **RTTI** for a very long time. **Delphi 2010** extended the **RTTI** system from not only published properties to all elements of a class: fields, properties, and methods could be examined using this "**Extended RTTI**", not just for published properties, but also for private, protected and public fields.
Additionally, it introduced **Attributes**:
a system to annotate the class and its fields, properties and methods with extra information.

Free Pascal introduced support for **Attributes** quite early on, but they were limited to published properties. It took more than 10 years to catch up and implement, and several more years to merge the support for **Extended RTTI** in the main development compiler. The largest part of the work has now come to an end, and extended **RTTI** and attributes can now be used.

❷ WHY EXTENDED RTTI?

Before diving into the details of the **RTTI Api** and the possibilities, why would you want to use **RTTI**?
One of the main uses of **RTTI** is streaming:
it allows you to examine a class and write the contents of this class to file or database.
Later the information in file or database can be read and used to reconstruct the object.
All this can be done by a standalone system, that does not need to know the details of the classes it writes or reads: it can use the **RTTI** to examine the classes and to manipulate the classes.

To make this more specific, take the **`TTimer`** class:

```
TComponent= class (TPersistent)
published

Property Name : string Read FName Write SetName;
end;

TTimer = class (TCustomTimer)
published
  property Enabled: Boolean read FEnabled write SetEnabled default True;
  property Interval: Cardinal read FInterval write SetInterval default 1000;
  property OnTimer: TNotifyEvent read FOnTimer write SetOnTimer;
  property OnStartTimer: TNotifyEvent read FOnStartTimer write FOnStartTimer;
  property OnStopTimer: TNotifyEvent read FOnStopTimer write FOnStopTimer;
end;
```

Through the **RTTI**, a streaming system knows that the `TTimer` class has 6 properties that it can stream.
It knows the type of the property (*for instance* `Cardinal` *for Interval*), how it
must be read (*directly from the field* `FInterval`) or written (*through the* `SetInterval` *method*),
and what the default value is (1000):
the default value is a way to optimize the writing by not writing the value of a property if it matches the default value.

The same mechanism allows the **property inspector** to show objects that it does not know.

The **RTTI** above had quite some limitations:
Only published properties can be examined. The type of information that could be published was also limited: classes descendent from `TPersistent`, ordinal types, set types (*up to 32 elements*) and `float` types.
Methods had to be published as well in order to be usable as event handlers.

So no records, or no arrays, no public or protected properties or simply fields. The absence of annotations meant also that the **streaming** system had no possibility to store extra information (*for instance an alternative name for a property*).
With the introduction of **Extended RTTI** and **Attributes**, these restrictions were lifted:
Information of any type can be examined, fields can be examined.
All visibilities (**`public, protected, private`**) are now subject to inspection.

❸ WHY ATTRIBUTES ?

To see why annotations can be useful, assume you have a external **REST** service which serves and consumes data through **JSON**. For example a **JSON** contact person object

```
{
"first-name" : "michael",
"last-name" : "Van Canneyt"
"date-of-birth" : "19700707"
}
```

If you wish to model this data in an object, you need to use pascal identifiers, and pascal types:

```
TPerson = record
firstname : string;
lastname : string;
birthdate : TDateTime;
end;
```

Two problems become immediately apparent: the fieldnames (keys) in the **JSON** are not valid pascal identifiers, so you need to map somehow the names used in **JSON** to the field-names and vice versa.
The date field needs to be read (*and written*) in a format which is not handled by a simple `StrToDate` or `DateToStr` method. If multiple **REST** resources are used, then you need a mapping between the class and the **REST** resource to be consumed.

One way of doing this is creating some data structure which contains the mapping and which provide the format. Another way would be to have all data structures descend from a basic class which provides some methods to create the mapping:

```
TPerson = Class
  class function ResourceName : String;
  class function FieldToJSONName(const aField : String): String;
  class function FormatDate(aDate : TDatetime) : String;
  firstname : string;
  lastname : string;
  birthdate : TDateTime;
end;
```

If you additionally wish to save the data into a database using some **ORM** technology, then it becomes even more complicated: The names of the database fields can also be different from the fields used in the class. A similar technique can be used to introduce a second mapping for the database;

```
TPerson = Class
 Class function TableName : string;
 class function FieldToJSONName(const aField : String): String;
 class function ResourceName : String;
 class function FieldToDBField(const aField : String): String;
 class function FormatDate(aDate : TDatetime) : String;
 firstname : string;
 lastname : string;
 birthdate : TDateTime;
end;
```

The result usually is a multitude of functions in the declaration (*and their implementation, of course*) to take care of the necessary mappings and provide information on how and what to stream.

Attributes can make this a lot simpler:
The same information can be provided with simple annotations on the fields themselves:

```
[Table('People')]
[Resource('/REST/Contact')]
TPerson = Record
   [DBKeyField]
   id : integer;

   [JSONKey('first-name')]
   [DBField('pe_name')]
   firstname : string;

   [JSONKey('last-name')]
   [DBField('pe_lastname')]
   lastname : string;

   [JSONKey('date-of-birth')]
   [DateFormat('yyyymmdd')]
   [DBField('pe_birthdate')]
   birthdate : TDateTime;
end;
```

The above uses 5 attributes:

`Table` specifies the database table in which to save the object.

`Resource` specifies the URL at which the resource can be retrieved

`JSONKey` specifies the key to use when reading/writing the JSON Object.

`DBField` specifies the database `fieldname` to use when loading/saving from/to the database.

`DateFormat` specifies the formatting used in JSON for a date.

A system that consumes **REST** services and a second system that loads/saves objects to database can inspect these attributes using **RTTI** and use them to load or save the objects in the correct location and with the correct format. In a realistic system, more attributes will be needed, of course.
If you have only 1 service to create and one database table to maintain, using attributes may be overkill: The whole operation to create **JSON** can then of course be done much simpler:

```
function TPerson.ToJSON : TJSONObject;
begin
Result:=TJSONObject.Create([
'id',id,
'first-name',firstname,
'last-name',LastName,
'date-of-birth',FormatDateTime('yyyymmdd',birthdate)
]);
end;
```

But with many tables and many **REST** resources to create or consume, the use of attributes avoids writing a lot of functions such as the above or even to provide the necessary metadata for the systems that interact with the **REST** service or database.
From the declaration of the object the programmer can also immediately see how the mappings are defined, without needing to dive into functions that provide the same information.
Of course, it can be argued that this meta-information should not be inside the object itself, that these mappings should be constructed outside the business objects: the above approach introduces a coupling between the used **JSON** streaming framework and the business objects.
Similarly for the ORM (Object-Relational Mapping: the framework to save objects to a database).
Additionally, it does not cover all possible use scenarios, for example what if an object needs to be loaded from one database and saved in another database with a different structure?
This questions are legitimate, but are the subject of a separate discussion:
here we just want to illustrate a possible use of attributes.

❹ THE SYSTEM.RTTI UNIT

The classical **RTTI** as it existed since **Delphi** 1 (*or* **Free Pascal**) could be examined through the routines in the `TypInfo` unit. These were low-level routines, requiring manually allocating memory and using pointers. With the introduction of **Extended RTTI**, a new **API** to consult and use the **RTTI** was also introduced:

The extended **`RTTI`** needs to be examined using the **`System.Rtti`** unit. It provides the same functionality as the **`TypInfo`** unit offered, but offers more convenient APIs: record, objects.

For every type and kind of data, the RTTI unit contains a dedicated object which represents the RTTI data for that type. Here are the main classes:

`TRttiType` This is the parent class for **all** types.

`TRttiInstanceType` Represents the RTTI for a **class** type.

`TRttiRecordType` Represents the RTTI for a **record**.

`TRttiField` Represents the RTTI for a **field** of a record or class.

`TRttiProperty` Represents the **RTTI** for a **property** of a record or class.

`TRttiMethod` Represents the **RTTI** for a **method** of a record or class.

These classes offer methods to find related **RTTI**: for a class, its list of fields, for a method, its parameters. Almost all classes have a method to get the **Attributes** associated with the data or type they represent.
The property and field classes have a method to set the value of the field or property, given an instance. The method classes can be invoked, allowing you to call any method without knowing the exact calling mechanism. To make all this possible without using the actual types, a `TValue` type has to be used: this new type is much like a variant, but offers an exacter type match than a variant. It it used to **`set/get`** property or field values, it is used to return values of functions you call and has to be used to supply parameter values for methods that need parameters.

The APIs in this unit use classes, but the lifetime of these classes are completely managed by the unit: there is no need to free them. To make this possible a `TRttiContext` must be used when calling the APIs of the Rtti unit: when the context is freed, all instances of RTTI classes that are no longer used will be released.
NOTE that in **Free Pascal** the routines in the Rtti unit are just a convenience layer on top of the routines in the TypInfo unit: everything that can be done with the **`Rtti`** unit can also be done with the routines in the TypInfo unit (*except the Invoke functionality to call a method*). In Delphi, this is not the case: some functionalities are only present in the **`Rtti`** unit. So if you have code that you want to run in **FPC** and in **Delphi**, you should stick to using the **`Rtti`** unit.

❺ USING THE EXTENDED RTTI AND ATTRIBUTES

So, how can we use the **Extended RTTI** and the **Attributes**?
We'll demonstrate this by expanding on the example given above:
We'll create some routines that make a JSON object from data in any record, after reading this record from the database.
We've seen that 5 attributes were used. We'll start by introducing the complete code that defines the person record.

```
unit contact;

  {$mode objfpc}
  {$H+}
  {$modeswitch prefixedattributes}
  {$modeswitch advancedrecords}

interface

uses
  Classes, SysUtils, rest.types, rttitojson.types, objtodb.types;

  {$RTTI EXPLICIT Fields[vcPublic]}
  Type
  [Table('People')]
  [Resource('/REST/Contact')]
  TPerson = Record
    [DBKeyField]
    id : integer;

    [JSONKey('first-name')]
    [DBField('pe_name')]
    firstname : string;

    [JSONKey('last-name')]
    [DBField('pe_lastname')]
    lastname : string;

    [JSONKey('date-of-birth')]
    [DateFormat('yyyymmdd')]
    [DBField('pe_birthdate')]
    birthdate : TDateTime;
  end;
```

An extra attribute (DBKeyField) is used to indicate the key field for the record.

The block with modeswitch directives instruct the **Free Pascal** compiler to allow prefixed attributes - this is needed because traditionally, **Free Pascal** uses the attribute notation to specify modifiers for procedures and functions. The second directive allows the use of advanced records.
The *{$RTTI }* directive deserves a mention: This directive controls how much **RTTI** is generated for your classes and records.

{$RTTI EXPLICIT FIELDS[vcPublic]}

The **`EXPLICIT`** keyword tells the compiler that what follows is the only kind of information that should be generated, and it should ignore the **RTTI** settings for parent classes.

Alternatively `INHERITED` can be used to indicate that what follows should be generated in addition to what information is used for the parent class. Finally, the `FIELDS` keyword tells the compiler what kind of information should be generated for fields. The other possibilities are `METHODS` and `PROPERTIES`, which are used to specify what RTTI should be generated for methods and properties, respectively. Finally a list of visibilities must be specified: the visibility sections for which RTTI information must be generated.
The above directive therefore tells the compiler that it should only generate RTTI for public fields. The following directive tells the compiler to generate all possible RTTI information

```
{$RTTI EXPLICIT
FIELDS      [ vcPrivate, vcProtected, vcPublic, vcPublished]
METHODS     [ vcPrivate, vcProtected, vcPublic, vcPublished]
PROPERTIES[ vcPrivate, vcProtected, vcPublic, vcPublished]
}
```

The next thing to note is the presence of the rest.types `rtitojson.types` and `objtodb.types` units. These units define the various attributes used in the definition. An attribute in Delphi is any class which descends from `TCustomAttribute`.
The attribute notation

```
[JSONKey('first-name')]
```

Tells the compiler that it should construct an attribute of class `JSONKey` or `JSONKeyAttribute`, (*the compiler will optionally strip off the Attribute from the classname when looking for the class*) and that it should pass the string 'first-name' to the constructor of the class.
The following unit defines the attributes for the JSON streaming:

```
unit rttitojson.types;
{$mode ObjFPC}{$H+}
interface

uses
  Classes, SysUtils;
Type
{ JSONKeyAttribute }
JSONKeyAttribute = class(TCustomAttribute)
private
  FKey: string;
public
  constructor create(aKey : String);
  property Key : string Read FKey;
end;
{ DateFormatAttribute }
DateFormatAttribute = class(TCustomAttribute)
  FFormat : String;
Public
  constructor create(aFormat : String);
  Property DateFormat : String Read FFormat;
end;
```

The implementation is quite simple:

```
implementation

{ JSONKeyAttribute }
constructor JSONKeyAttribute.create(aKey: String);
begin
  FKey:=aKey;
end;

{ DateFormatAttribute }
constructor DateFormatAttribute.create(aFormat: String);
begin
  FFormat:=aFormat;
end;

end.
```

The `GetAttributes` call can be used to get all attributes on an identifier. The `GetAttributes` call can be used to get a single attribute, by specifying the class for the attribute.

❻ USING ATTRIBUTES TO CREATE JSON

How can we use this to convert a record to a JSON object ? This is done with a `TRTTIJSONWriter` in a separate unit (we could also have used a class obviously).
Note that the TPerson record in no way references the writer: only the attributes are needed to define our TPerson record.

```
unit rttitojson.writer;
{$mode ObjFPC}
{$H+}
{$modeswitch advancedrecords}
interface

uses
   Classes, SysUtils, fpJSON, typinfo, rtti, rttitojson.types;

Type
   TRTTIJSONWriter = record
private
   Ctx : TRTTIContext;
   function DateFieldToJSON(aDate: TDateTime; Fld: TRTTIField): TJSONData;
   function FieldToJSON(aData: Pointer; Fld: TRTTIField): TJSONData;
Public
   Procedure ToJSON(aData : Pointer; aTypeInfo : PTypeInfo; aJSON : TJSONObject);
   Generic function ToJSON<T>(var aData : T) : TJSONObject;
   class function create : TRTTIJSONWriter; static;
   procedure Free;
end;
```

The create and free methods create and free a `TRttiContext` which is needed to call the various functions from the Rtti unit:

```
class function TRTTIJSONWriter.create: TRTTIJSONWriter;
begin
   Result.Ctx:=TRttiContext.Create(False);
end;

procedure TRTTIJSONWriter.Free;
begin
   Ctx.Free;
end;
```

The main entry point is the generic `ToJSON` function. In Free Pascal notation:

```
 Generic function ToJSON<T>(var aData : T) : TJSONObject;
```

This is actually just a convenience function with a quite simple implementation. It creates the result JSON Object and calls an overloaded version of the `ToJSON` function, passing it the type information of the type T (a record):

```
generic function TRTTIJSONWriter.ToJSON<T>(var aData : T) : TJSONObject;
begin
  Result:=TJSONObject.Create;
  try
    ToJSON(@aData,TypeInfo(T),Result);
  except
    Result.Free;
    Raise;
  end;
end;
```

The actual work is done in the other ToJSON function.

In essence this gets the **Rtti** for the record and loops over all fields:
the list of fields is retrieved with the GetFields call, which returns an array of TRttiField instances. For each field it determines the key name to use when writing the **JSON**, and then calls FieldToJSON to convert the value of the field to a **JSON** data element, which is then added to the JSON object.

```
procedure TRTTIJSONWriter.ToJSON(aData: Pointer; aTypeInfo: PTypeInfo;
                                                    aJSON: TJSONObject);
Var
  R   : TRttiRecordType;
  Fld : TRttiField;
  Key : JSONKeyAttribute;
  KeyName : String;
begin
  R:=Ctx.GetType(aTypeInfo) as TRttiRecordType;
  For Fld in R.GetFields do
    begin
      KeyName:=Fld.Name;
      Key:=JSONKeyAttribute(Fld.GetAttribute(JSONKeyAttribute));
      if Assigned(Key) then
        KeyName:=Key.Key;
      aJSON.Add(KeyName,FieldToJSON(aData,Fld));
    end;
end;
```

NOTE the use of the GetAttribute call with the JSONKeyAttribute class name. If no attribute of this class exists for the field, Nil will be returned, and in that case the actual field name is used. The FieldToJSON call starts by getting the value of the field using the GetValue name, passing the function a pointer to the record. The GetValue function will use the field's **RTTI** to calculate the correct memory address to retrieve the value:

```
function TRTTIJSONWriter.FieldToJSON(aData: Pointer; Fld : TRTTIField): TJSONData;
var
  V : TValue;
begin
  V:=Fld.GetValue(aData);
  case V.Kind of
    tkFloat : if V.TypeInfo=TypeInfo(TDateTime) then
                Result:=DateFieldToJSON(V.AsDateTime,Fld)
              else
               Result:=TJSONFloatNumber.Create(V.AsDouble);
    tkInt64 : Result:=TJSONInt64Number.Create(V.AsInt64);
    tkInteger : Result:=TJSONIntegerNumber.Create(V.AsInteger);
    tkEnumeration : Result:=TJSONString.Create(GetEnumName(V.TypeInfo,V.AsInteger));
    tkAString,
    tkString : Result:=TJSONString.Create(V.AsString);
    tkWString,
    tkUString : Result:=TJSONString.Create(UTF8Encode(V.AsUnicodeString));
  else
    Raise Exception.Create('Unsupported type');
  end;
end;
```

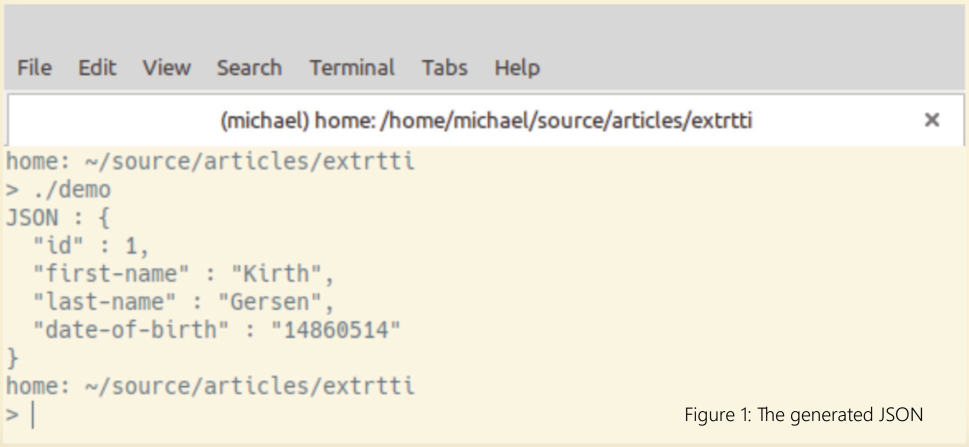
After the value was retrieved, the type of the value is examined, and an appropriate **JSON** data structure is created using the data. For a TDateTime field, a special routine is called which uses the DateFormat attribute to create a correctly formed **JSON** string:

```pascal
function TRTTIJSONWriter.DateFieldToJSON(aDate : TDateTime; Fld : TRTTIField) : TJSONData;
var
  Res,Fmt : string;
  DateFormat : DateFormatAttribute;
begin
  Fmt:='';
  DateFormat:=DateFormatAttribute(Fld.GetAttribute(DateFormatAttribute));
  if Assigned(DateFormat) then
    Fmt:=DateFormat.DateFormat;
  if Fmt='' then
    Res:=DateToISO8601(aDate)
  else
    Res:=FormatDateTime(Fmt,aDate);
  Result:=TJSONString.Create(Res);
end;
```

With this, we're all done. The following program tests our code:

```pascal
uses sysutils, contact, fpjson, rttitojson.types, rttitojson.writer;

Procedure WriteJSON(P : TPerson);
var
  Writer   : TRTTIJSONWriter;
  JSON     : TJSONObject;
begin
  JSON     := Nil;
  Writer   := TRTTIJSONWriter.Create;
  try
    JSON   :=Writer.ToJSON<TPerson>(P);
    Writeln('JSON : ',JSON.FormatJSON());
  finally
    JSON.Free;
    Writer.Free;
  end;
end;

var Person : TPerson;

begin
  With Person do
    begin
      id:=1;
      FirstName:='Kirth';
      LastName:='Gersen';
      birthdate:=EncodeDate(1486,5,14);
    end;
  WriteJSON(Person);
end.
```

The output is shown in figure 1 on page 9 of this article.


File Edit View Search Terminal Tabs Help
(michael) home: /home/michael/source/articles/extrtti

```
home: ~/source/articles/extrtti
> ./demo
JSON : {
  "id" : 1,
  "first-name" : "Kirth",
  "last-name" : "Gersen",
  "date-of-birth" : "14860514"
}
home: ~/source/articles/extrtti
> |
```

Figure 1: The generated JSON

❼ USING ATTRIBUTES TO READ AND WRITE FROM DATABASE

A similar unit with a similar record can be created for the reading/writing of an object from database:

```
TRTTIDBReader = record
private
  Ctx : TRTTIContext;
  Conn : TSQLConnection;
  function CreateQuery(aSQL : String) : TSQLQuery;
  Procedure DBFieldToField(DBData : TDataset; aData: Pointer; Fld: TRTTIField);
  Procedure FieldsToParams(Params : TParams; aData: Pointer; Rec : RttiRecordType);
  function RecordToWhereClause(Rec: TRttiRecordType): String;
  Function RecordToSQL(Rec: TRttiRecordType) : String;
  procedure ValueToParam(V: TValue; Parm: TParam);
Public
  function ReadFromDB(aData : Pointer; aTypeInfo: PTypeInfo) : Boolean;
  Generic function ReadFromDB<T>(var aData : T) : Boolean;
  class function create(aConnection : TSQLConnection): TRTTIDBReader; static;
  procedure Free;
end;
```

We'll examine only the main methods. The `generic` function is again a small convenience wrapper:

```
Generic function TRTTIDBReader.ReadFromDB<T>(var aData : T) : Boolean;
begin
   Result:=ReadFromDB(@aData,TypeInfo(T));
end;
```

The actual work is done in the following function. It starts by using the rtti of the record to create an SQL `SELECT` statement, which is then used to create a query object: the query is opened, and if it returns a result, the record is loaded from the result:

```
Function TRTTIDBReader.ReadFromDB(aData : Pointer; aTypeInfo: PTypeInfo) : Boolean;
var
  lRtti : TRttiRecordType;
  Qry   : TSQLQuery;
  Fld   : TRttiField;
  SQL   : String;
begin
  lRtti := Ctx.GetType(aTypeInfo) as TRttiRecordType;
  SQL   := RecordToSQL(lRtti);
  Qry   := CreateQuery(SQL);
  try
    FieldsToParams(Qry.Params,aData,lRtti);
    Qry.Open;
    Result  := not Qry.IsEmpty;
    if Result then
      For Fld in lRtti.GetFields do
         DBFieldToField(Qry,aData,Fld);
    finally
    Qry.Free;
  end;
end;
```

The main loop here is again a loop over the fields in the record. The `DBFieldToField` procedure transfers the data from the dataset fields to the fields in the record.

The `RecordToSQL` routine constructs an SQL `Select` statement to retrieve the data for the record from the database. It uses the `Table` attribute on the record definition to determine the table name, and uses the `DBField` attribute on the fields of the record definition to get the name of the database table fields. Both times, the pascal name of the record type or field is used as a fallback:

```
Function TRTTIDBReader.RecordToSQL(Rec: TRttiRecordType) : String;
var
  Tbl : TableAttribute;
  Fld : TRttiField;
  DBField : DBFieldAttribute;
  FieldName : String;
  TableName : string;

begin
  Result:='';
  For Fld in Rec.GetFields do
    begin
      FieldName:=Fld.Name;
      DBField:=DBFieldAttribute(Fld.GetAttribute(DBFieldAttribute));
      if Assigned(DBField) then
        FieldName:=DBField.Name;
      if Result<>'' then
        Result:=Result+', ';
      Result:=Result+FieldName;
    end;
  TableName:=Rec.Name;
  Tbl:=TableAttribute(Rec.GetAttribute(TableAttribute));
  If Assigned(Tbl) then
    TableName:=Tbl.Table;
  Result:='SELECT '+Result+' FROM '+TableName;
  Result:=Result+' WHERE '+RecordToWhereClause(Rec);
end;
```

The `RecordToWhereClause` function executes a similar loop to construct the where clause that will select a single record from the database, based on the key fields, which are marked with the `DBKeyField` attribute:

```
on TRTTIDBReader.RecordToWhereClause(Rec: TRttiRecordType) : String;
var
  Fld : TRttiField;
  DBField : DBFieldAttribute;
  FieldName : string;
begin
  Result:='';
  For Fld in Rec.GetFields do
  begin
    FieldName:=Fld.Name;
    if Assigned(Fld.GetAttribute(DBKeyFieldAttribute)) then
    begin
      DBField:=DBFieldAttribute(Fld.GetAttribute(DBFieldAttribute));
      if Assigned(DBField) then
        FieldName:=DBField.Name;
      if Result<>'' then
        Result:=Result+' AND ';
      Result:=Result+'('+ FieldName+' = :'+FieldName+')';
    end;
  end;
end;
```

The record definition presented earlier reads records from the `People` table, which can be defined in SQL as follows:

```
create table people (
id int not null,
pe_name varchar(127) not null,
pe_lastname varchar(127) not null,
pe_birthdate date not null
);
```

The above routines will create the following SQL:

```
SELECT id, pe_name, pe_lastname, pe_birthdate
FROM people WHERE (id = :id)
```

To fill the parameters used in the WHERE clause, the following routine is used, which is just another variation on the previous routines:

```
Procedure TRTTIDBReader.FieldsToParams(Params: TParams; aData: Pointer;
                                        Rec: TRTTIRecordType);
var
  Fld       : TRttiField;
  DBField   : DBFieldAttribute;
  Parm      : TParam;
  V         : TValue;
  FieldName : string;

begin
  For Fld in Rec.GetFields do
    begin
      FieldName:=Fld.Name;
        if Assigned(Fld.GetAttribute(DBKeyFieldAttribute)) then
          begin
            DBField:=DBFieldAttribute(Fld.GetAttribute(DBFieldAttribute));
            if Assigned(DBField) then
               FieldName:=DBField.Name;
            Parm:=Params.FindParam(FieldName);
            if Assigned(Parm) then
            begin
               V:=Fld.GetValue(aData);
               ValueToParam(V,Parm);
            end;
          end;
    end;
end;
```

The `ValueToParam` routine applies the `TValue` to a `TParam`. No attributes are used this time, the TValue contains all information we need to apply the value to a query`parameter

```
Procedure TRTTIDBReader.ValueToParam(V : TValue; Parm : TParam);
begin
  Case V.Kind of
    tkInteger : Parm.AsInteger:=V.AsInteger;
    tkInt64 : Parm.AsInteger:=V.AsInt64;
    tkAString,
    tkString : Parm.AsString:=V.AsString;
    tkUString : Parm.AsUnicodeString:=V.AsUnicodeString;
    tkWString : Parm.AsUnicodeString:=V.AsUnicodeString;
    tkFloat :
      if V.TypeInfo=TypeInfo(TDateTime) then
        Parm.AsDateTime:=V.AsDateTime
      else
        Parm.AsFloat:=V.AsDouble;
  else
    Raise Exception.Create('Unsupported type')
  end;
end;
```

Finally, the `DBFieldToField` routine is used to write the content of a database field to a field in the record. The `TRttiField.SetValue` method accepts a `TValue`, so the routine starts by creating such a value from the database field:

```pascal
Procedure TRTTIDBReader.DBFieldToField(DBData : TDataset; aData: Pointer;
        Fld: TRTTIField);
var
  V : TValue;
  DBField : DBFieldAttribute;
  DBFld : TField;
  FieldName, S : String;
  DT : TDateTime;
  U : UnicodeString;
begin
  FieldName:=Fld.Name;
  DBField:=DBFieldAttribute(Fld.GetAttribute(DBFieldAttribute));
  if Assigned(DBField) then
    FieldName:=DBField.Name;
  DBFld:=DBData.FieldByName(FieldName);
  Case DBFld.DataType of
    ftInteger,
    ftSmallint,
    ftWord : TValue.Make(DBFld.AsInteger,fld.FieldType.Handle,V);
    ftLargeInt : TValue.Make(DBFld.AsLargeInt,fld.FieldType.Handle,V);
    ftString :
    begin
      S:=DBFld.AsString;
      TValue.Make(@S,fld.FieldType.Handle,V);
    end;
    ftWideString:
    begin
      U:=DBFld.AsUnicodeString;
      TValue.Make(@U,fld.FieldType.Handle,V);
    end;
    ftDate
    ftTime,
    ftDateTime:
      begin
        DT:=DBFld.AsDateTime;
        TValue.Make(@DT,fld.FieldType.Handle,V);
      end;
  else
   Raise Exception.Create('Unknown field type: '+GetEnumName(TypeInfo(TFieldType),
  end;
  Fld.SetValue(aData,V);
end;
```

Once the value is created, the Fld.SetValue copies the value to the appropriate record field. With all this in place, the code to read a record from a database and write it out as a JSON structure is as follows:

```
Function LoadPerson(aID : Integer) : TPerson;
var Reader : TRTTIDBReader;
    Conn    : TSQLConnection;
    P       : TPerson;
    ReadOK  : Boolean;
begin
  P.ID:=aID;
  conn:=GetConnection;
  try
    Reader:=TRTTIDBReader.Create(Conn);
    ReadOK:=Reader.ReadFromDB<TPerson>(P);
    if not ReadOK then
     Writeln('Failed to read person');
     Result:=P;
  finally
    Conn.Free;
  end;
end;

begin
  WriteJSON(LoadPerson(1));
end;
```

The same code could then be used to load and write all other objects. If we insert a record in the database with the following SQL:

```
;
insert into people values (1,'Kirth','Gersen','1486-05-14');
```

Then the output of our modified program will look exactly the same as figure 1 on page 9.

❽ CONCLUSION

The arrival of **Extended RTTI** in free pascal opens up new possibilities in **Free Pascal** and Lazarus: as shown here, **it is possible to create a functional JSON serialization** and a mini **ORM** framework with very little code. Needless to say, in a real world program some more attributes will be needed, but the basic working would be as described here. Not in the least, the arrival of **Extended RTTI** in **Free Pascal** will enable the use in **Free Pascal** of similar such frameworks written for **Delphi**.

THE ARRIVAL OF
EXTENDED RTTI
IN FREE PASCAL
DOES ENABLE THE USE
OF FRAMEWORKS
WRITTEN FOR DELPHI.

ADVERTISEMENT

BLAISE PASCAL MAGAZINE 116

Multiplatform / Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js
Databases / CSS Styles / Progressive WebApps
Android / IOS / Mac Windows & Linux

SUBSCRIPTION
1 YEAR
€ 64,22 EX VAT
€ 70 INC VAT

https://www.blaisepascalmagazine.eu/en/register/

JUN 13-14 2024 | AMSTERDAM

Spinnekop 3, 1444 GN Purmerend, the Netherlands

Delphi Summit

WHEN WE WORKED FROM HOME

BY IAN BARKER, EMBARCADERO DEVELOPER ADVOCATE.

The global pandemic of 2020 was something that did so much damage. Lives lost, livelihoods lost, in fact, a whole lot of loss. It's going to be written about in future history books, for sure. Life changed too in a myriad of other ways. Many companies embraced the concept of remote and work from home scenarios. Some out of a greed, or need, to keep making money while the world learned that coughing hands that touch fruit touch you, touch your workers, and eventually touch your finances. The majority, of course, simply heeded scientific advice to stay away from each other, shun crowded public spaces, and spurn the daily commute.

As COVID fades into history there's a bit of reversal going on, particularly in the tech industry, with 'adjustments to headcounts' en masse. Many tech giants who had gushed worthily over allowing staff to work from home, either permanently, or at least primarily, have since reneged on that idea and have faltered on the altar of a corporate fundamental belief that in order to be efficient it's better to lump as many people as possible into a single physical space - the office.

The bigger the company the more likely they are to have a huge whale-like building which is packed to the gills with people who could have regained a couple of hours of their personal time by not being there and instead attend meetings virtually. It seems some lessons learned during The Pandemic were very quickly forgotten.

SHAREHOLDERS SAY IN-PERSON IS BEST

If you are a software developer I guarantee you that you are no more efficient by having to commute to an office and spend hours wedged in a cubicle on the off-chance that a meeting might need to happen where your physical presence can in some way improve it. I say this as a senior developer with nearly 40 years of experience at every single level for all sorts of companies from tiny start-ups to Fortune 500 sprawling giants. Funny how the decision 'in the office not the home' is often made by leaders who do not actually come into the building where the great unwashed ply their trade or, indeed have what anyone would recognize as a regular office job.

With the advent of capable meeting software such as Google Meet, Skype, Microsoft Teams, Zoom, GoToMeeting and the many other competitors, combined with affordable high-quality webcams and mics or headsets there is almost nothing that requires you to be physically present if you are a software developer, designer, project manager, or QA. You're not a baker, you do not need to physically smell your products. You're not a winemaker so there is no need to tramp up and down acres of vines tutting loudly at soil moistures and the friability of your terroir. Your job literally exists in the ether. You ARE The Matrix. That's your proper place.

There are plenty of options too if it's a matter of trust. Lots of apps which will assess your worker's activity, check they were not goofing off or 'Netflix and chilling' when they should have been bug fixing. If you need to be a control freak geek boss there is a whole industry of tattle tale software apps vying to be your number one spy master.

SOMETIMES YOU JUST NEED SWAG AND TO HANG OUT

But some things are better done in person and one of those got decimated by The Pandemic - the in-person tech conference. You can attend any kind of virtual event you like but I can tell you for sure that the casual connections you make with like-minded techies at play are much more profound and long-lasting if they are in-person. Granted, as a species we geeks and nerds do have a bit of a reputation for poor personal hygiene and a dress sense that could blind a unicorn at 100 metres but I think that's mostly an undeserved thing...mostly.

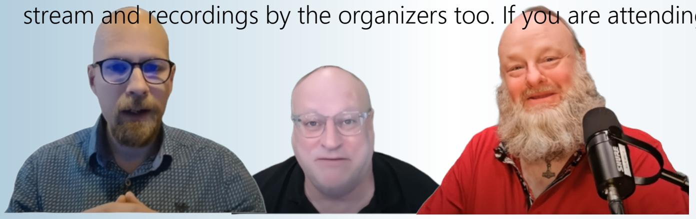
As the world finally got on top of COVID we all returned to worrying about regular things like nuclear war wiping us out instead of the unwashed handles of a grocery store door. Little by little the in-person events have returned. Last October I got to go to the Foren Tage event in Germany. Since then there has been a steady uptick in face-to-face meetings and conferences.

THE GLOBAL DELPHI SUMMIT 2024

If you can make it to Amsterdam on JUNE 13TH AND 14TH then I absolutely recommend you go to the Global Delphi Summit organized by GDK Software and various sponsors including the ever-wonderful Barnsten, and, of course, Embarcadero.

I will be there for both days, all day. I am presenting keynotes on day one and day two but I'll also be hanging out, meeting with you all and answering your questions where I can. The unmissable Marco Cantú is presenting a special session and other well-known Delphi folk such as Dr Bob will be around too. It's going to be truly awesome.

I realize not everyone will be able to make it. I know how that feels, I've been in that position many times too, and it's painful. I will be streaming live throughout both days, and there will be an official stream and recordings by the organizers too. If you are attending in June, come say hi.



Delphi Summit
JUN 13-14 2024 | AMSTERDAM

Delphi Summit
JUN 13-14 2024 | AMSTERDAM

embarcadero®
Official Technology Partner

For just 249,-* you will get:

- Admission for both days, with all speakers
- Each day from 10:00 a.m. to 6:00 p.m.
- Lunch and drinks included
- Free video recording of all sessions
- All sessions in English
- Free parking + easy public transport from Amsterdam Schiphol Airport
- Hotel rooms can be booked separately after ordering
- Discount with two or more tickets

***Early bird discount, ends April 1st**

https://delphisummit.com

Extra 10% discount for Blaise Pascal readers! Use code AVB16BT6S1BF at the check-out

See you there!

Delphi Summit
JUN 13-14 2024 | AMSTERDAM

embarcadero
Official Technology Partner

Delphi Summit 2024

30+ **In Depth Sessions**
all about Delphi and Pascal

300 **Attendees**
meeting fellow developers

2 **Full Delphi Days**
from 10 PM to 6 PM

15+ **Acclaimed speakers**
from all over the world

THE DELPHI SUMMIT IN A NUTSHELL

About the summit.

The summit is organised by GDK Software, in partnership with Embarcadero and Barnsten. It is a two-day event packed with the latest and greatest news and innovations, all about our favourite programming language: Delphi.

LOCATION AND SPEAKERS

Meet and great

The location is the H20 Esports conference centre, located near Amsterdam, The Netherlands. Speakers include Jim McKeeth, Ian Barker, Marco Cantu and many more!

Join the Delphi Summit: **https://delphisummit.com**

Delphi Summit 2024

Agenda

Amsterdam. June 13-14, 2024

Day 1: Thursday 13th

09:00	**Registration**		
	Main stage		
10:00	**Welcome and opening – Kees de Kraker and Marco Geuze**		
10:15	**Keynote – Jim McKeeth and Ian Barker**		
11:00	Coffee Break & Network		
	Stage Sydney	**Stage Alexandria**	**Stage Rio**
11:30	**Cary Jensen** - Selected Advanced FireDAC Technologies FireDAC supports a wide range of powerful and useful operations. This session will discuss and demonstrate four of the more interesting ones, including caching updates, batch move operations, using FireDAC built-in functions, and Local SQL.	**Fabrizio Bitti -** Creating a real-life Blockchain with Delphi Demonstrate how a blockchain works and what it is used for. All with Delphi in a multithread environment to mine the blocks.	**Steffen Nyeland -** I can, therefore IAM Changing your application login process to an IAM (Identity and Access Management) controlled process
12:30	Lunch, Network & Gaming		
13:30	**Marco Cantu** - Building FireMonkey Apps with Style Unlike VCL, styles in FireMonkey don't only determine the graphical elements of a UI control, but also its architecture. In this session, we'll explore how styles work, how to customize controls at runtime, how to build new styled FMXcomponents, and how this all helps building a single-source multi-device UI.	**Richard Hatherall –** AWS SDK	**Serge Pilko** – How to replace DataBase components with Rest API calls in Delphi An introduction to REST and creating a cross-platform REST Client application, using Embarcadero's REST Client library to replace database access components.
14:30	**Jim McKeeth -** Evidence Based Delphi Engineering Why do you write code *that way*? Chances are it is “the way you've always done it.” Learn how to gather the evidence you need to know the *right way*.	**Frank Lauter** - MVVM - The Delphi Way! A waste of time or a way to keep the source code maintainable? Frank Lauter will present his view on the MVVM pattern and explain which steps are necessary for new and legacy applications.	**Primož Gabrijelčič** - Defensive programming Learn from someone with 35 years of experience how to write code that will be easy to understand now, and in the future. Dive into some of my own ,laughable, terrible code examples with me and get easy-to-reuse advice on how to improve
15:30	Break & Network		
16:00	**Marco Geuze** – Delphi and AI Large language models (LLMs) provide significant help for development. Learn how to use a private LLM in Delphi without giving away your privacy and source code.	**Bob Swart -** REST with WebBroker in Delphi	**Conrad Vermeulen** - From monoliths to microservices Building composable applications with a bit of runtime config
17:00	Network & Gaming		
18:00	Day 1 ends		

Day 2: Friday 14th

09:00	**Registration**		
	Mainstage		
10:00	**Welcome and opening – Kees de Kraker and Marco Geuze**		
10:15	**Panel discussion with Jim McKeeth, Marco Cantu, Ian Barker and MVPs**		
11:00	Coffee Break & Network		
	Stage Sydney	Stage Alexandria	Stage Rio
11:30	**Kees de Kraker** – Test-driven development Using the TTD approach in a monolith application is not feasible, right? Join this session to discover the impossible.	**Patrick Quist** – Linux Delphi Services A journey through the Cloud(s)	**Stefan Glienke** – Spring4D Some goodies from the Spring4D collections.
12:30	Lunch, Network & Gaming		
13:30	**Olaf Monien - REST** Easy Connecting to REST APIs and visualizing data on desktop and mobile devices.	**Christoph Schneider** – Firestone Cloud For the Firestore Cloud database, the FB4D open-source library contains everything you need to access it from VCL/FMX applications. In this session, the author will show you how easy it is to write and read a document and to be notified of changes in the database with the new object-to-document wrapper.	**Patrick Prémartin** – Our users want to access their data from anywhere, on any type of device, with or without an Internet connection. Some also want to work together offline or online, remotely or on-site, on desktops, laptops, smartphones or tablets. Here's an easy-to-implement solution in Delphi to transform any local database into a synchronized one
14:30	**Carlos Agnes** - The Best of Delphi Underground A set of small Delphi secrets and how they work under the hood. IDE and debugging tips, historical issues like why the base date for TDateTime is 12/30/1899, Exceptions stacks, interface tricks, and the dictionary of secrets.	**Andrea Raimondi** - Algorithmic password hardening From the forgotten lessons of Enigma to generating salts and scrambling passwords, Andrea will guide you through the best ways to keep everything safe.	**Ray Konopka** - Component Building: Fundamentals This session focuses on the fundamental techniques required for building robust Delphi components. We build a custom component, showing the key classes from which all components descend, followed by an analysis of the anatomy of a component. We conclude with a discussion on the proper way to distribute custom components through runtime and design packages.
15:30	Break & Network		
15:45	To be announced		
	Mainstage		
16:15	**Ian Barker -** What to do if you're old, ugly, and everything is annoying Join Ian for this session where he applies his uniquely lively style of presentation to the subject of software development in an age where everyone wants your apps to be free, have a name like ZZxQFlmbl, and be 'monetized' by a YouTube influencer with green hair, a pierced fingernail, and their own brand of hair removal creme.		
16:45	**Door prize giveaway**		
17:00	**Closing talk with Jim McKeeth, Kees de Kraker and Marco Geuze**		
17:15	Network & Gaming		
18:00	Conference ends		

embarcadero®
TECHNOLOGY PARTNER

GDK software

Starter Expert

WHEN THE TIME IS RIGHT

We have looked at breakpoints before. They give us the ability to run our app and break at any suspicious code that we then want to inspect under the debugger.

But what if that code is in a loop, or called from within a loop?
The breakpoint will be hit in each iteration of the loop and the error may only happen after a great many iterations. That could be hundreds of times for which we have to just press F9 to continue. There are better ways to deal with that. We can tell the debugger when it should and when it should not pause the app at a breakpoint.

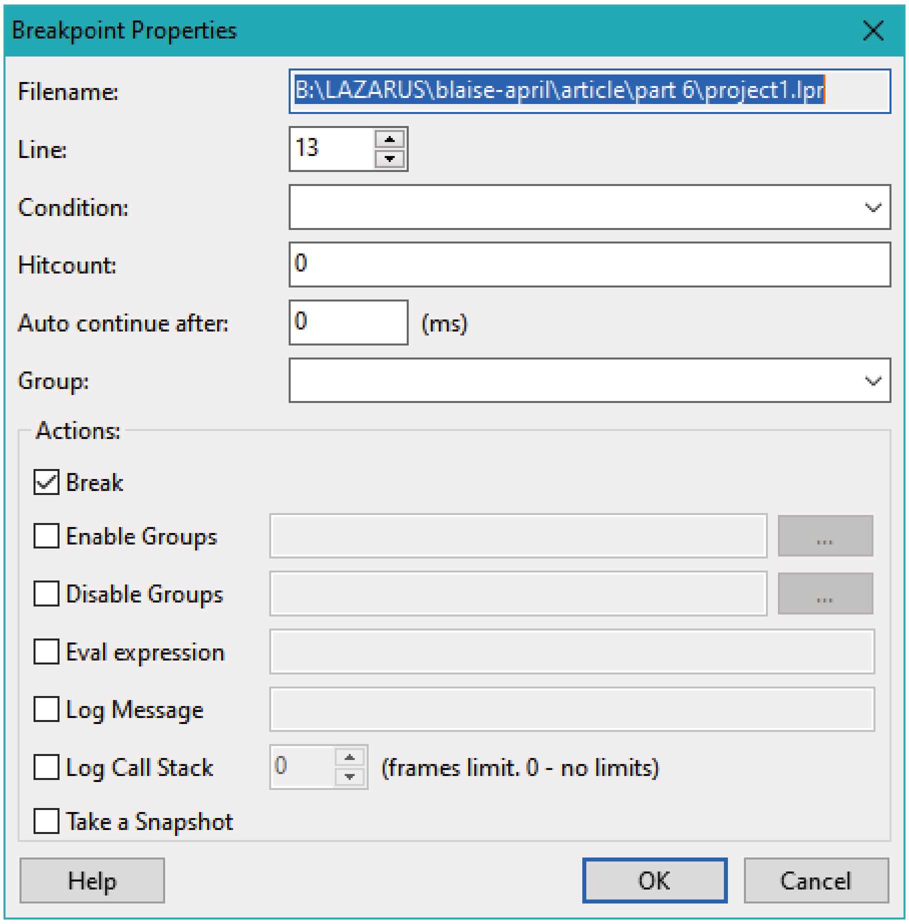
The following app converts Hex numbers to integer, and we test it by calling the method with all hex numbers from **`$000`** to **`$FFF`**. The result must be the same as FPC's build in "`StrToInt`".

```pascal
program project1;
{$Mode objfpc}{$H+}

uses SysUtils;

function Hex2Int(h: string): integer;
const
  c = '0123456789ABCDEF';
  d: array [0..3] of integer = (1, 16, 265, 4096);
var
  i, j: Integer;
begin
  Result := 0;
  for i := 0 to Length(h)-1 do begin
    j := pos(h[Length(h)-i], c) - 1;
    Result := Result + j * d[i];
  end;
end;

procedure Test;
var
  t,t1,t2,t3: String;
  i: Integer;
begin
  for t1 in ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'] do
  for t2 in ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'] do
  for t3 in ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'] do
  begin
    t := t1+t2+t3;
    i := Hex2Int(t);
    WriteLn('$'+t, ' = ', i);
    if i <> StrToInt('$'+t) then
      exit;
  end;
end;

begin
  Test;
  readln;
end.
```

The app will print a lot of lines and then stop with

```
$0FE = 254
$0FF = 255
$100 = 265
```

It should have run up to $FFF, but our Hex2Int returned the wrong value for $100.
Simply setting a breakpoint in Hex2Int will require us to continue 256 times before we get to that value. So we need some other approach. The sample app of course is very simple, and we could just start the loop at the value we need. But in a real life app, we may not have that option. So lets pretend we have to run through all the iterations before we can debug the issue.
Following are 3 examples showing different ways to stop only in the buggy iteration. Each uses a different property of the breakpoint. The properties can all be set in the following dialog:



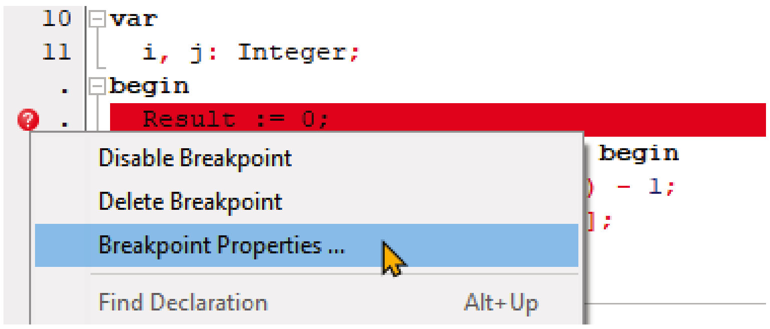
Breakpoint Properties

Filename:	B:\LAZARUS\blaise-april\article\part 6\project1.lpr
Line:	13
Condition:	
Hitcount:	0
Auto continue after:	0 (ms)
Group:	

Actions:

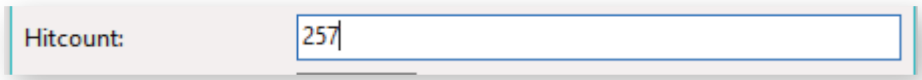
- [✓] Break
- [ ] Enable Groups
- [ ] Disable Groups
- [ ] Eval expression
- [ ] Log Message
- [ ] Log Call Stack 0 (frames limit. 0 - no limits)
- [ ] Take a Snapshot

The dialog itself can be opened from the context menu of each breakpoint in the editor, or from the toolbar of the breakpoint window.



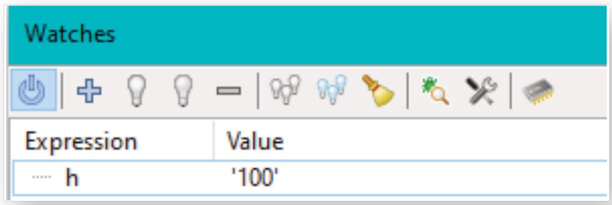
```
var
  i, j: Integer;
begin
  Result := 0;
```

Option 1: Hit Count
In the example we can make use of the knowledge that it will happen in the `257th` iteration. (*For cases were this isn't known upfront, see the paragraph on "pass count" to find out how to get that number*)
We will set a breakpoint at the start of **`Hex2Int`** on line **`13`**. Then we go to the properties and set "**Hit Count**" to 257.



Hitcount: 257

This will skip the first 256 hits of the breakpoint, skipping the calculation for the values 0 to 255. Running the application will take us to the breakpoint, we can evaluate "h" to see it contains the hex string `100`.



Watches

Expression	Value
h	'100'

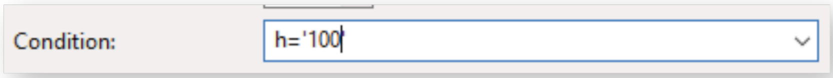
Now, we can fallback to the methods we have used in some of the previous articles.
Using stepping and watches we will find that it works fine for the lower 2 digits (*both zero*) at the end, but then multiplies the "1" with 265. A simple typographical error, easy to fix. It should be 256.

```
9.    d: array [0..3] of integer = (1, 16, 256, 4096);
```

After that, the application runs to the end, as it should.

> **NOTE**: For backends other than **FpDebug** the hit-count may be zero or one-based. It may need to be adjusted accordingly. Hit-Count is handled directly by the backend (**gdb, lldb or FpDebug**).

Option 2: Condition
An alternative approach is to use a condition in the breakpoint. Then we don't need to know how often it may be hit. All we need to have is the value of some variable or expression. In our case we know that "h" must be '100'. We enter this in the properties:



Condition: h='100'

Running the application with this breakpoint will take us to the same call of "**`Hex2Int`**", with the parameter "h" = '100'.

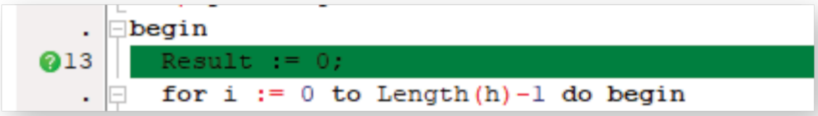
> **NOTE**: Conditions in breakpoint may work different depending on the debugger backend.
> With **FpDebug** any Pascal expression should work. With other backends (**gdb or lldb**) the condition needs to be supported by each of those backends.
> Comparing strings does not work with either of them.
> Also depending on the backend, if an expression can not be computed, or does not return a boolean value, the debugger may either always stop (*ignore the expression*), or never stop (*treat it as false*). **FpDebug** will ignore such expressions.

In our case, all we needed was the condition. We reached the code in the state we were interested in, on the very first hit for which the condition was true. In other cases conditions may only test some aspect of the required state, and the breakpoint may still stop a couple of times before the bug is reached. For this, conditions and hit-count can be combined.
Under **FpDebug** the hit count will be applied only against hits for which the condition was true.
So you could check if the first last in "h" was '0' by setting the condition: **`h[3]='0'`** and the **Hit-Count=17**, which would stop the 17th time that "h" ends with a '0'.

Option 3: Let the caller decide

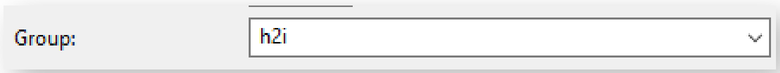
Sometimes, we don't have a good condition to test for when the issue will arise. But we may know it always happens when certain criteria are met in the caller or in some other code that is run before the bug manifests. In such a case we need a breakpoint in that other code (*in this example in the caller*) to evaluate the condition.

For this we need 2 breakpoints. The first one will be on **line 13** as before. However, this time we will set this breakpoint as disabled.



```
begin
13    Result := 0;
      for i := 0 to Length(h)-1 do begin
```

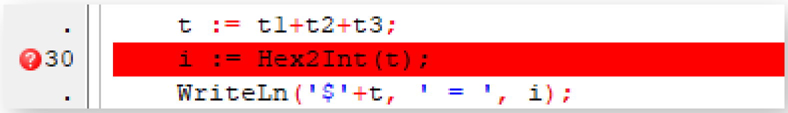
This breakpoint will not have a condition, nor will it have a Hit-Count. But we will assign it to a group. We will use "**h2i**" as group name, so we can easily identify it as the group for the function "**Hex2Int**".



Group: h2i

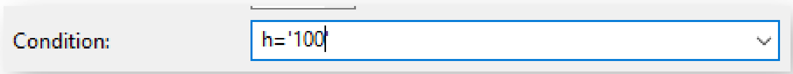
The second breakpoint goes into the caller. We will use it to hold the condition, and then to activate the first breakpoint once the condition is met. That way, the app will eventually stop at the first breakpoint, yet we can control when that will happen.

In the example we place that breakpoint at **line 30**.



```
      t := t1+t2+t3;
30    i := Hex2Int(t);
      WriteLn('$'+t, ' = ', i);
```

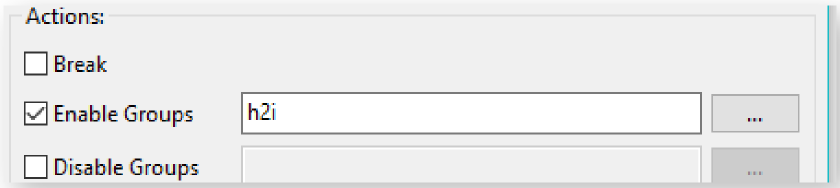
We need to change a few of its properties. First we need to set a condition. After all it is in the loop, and we don't want anything to happen until we got the right data for the call that we want to debug.



Condition: h='100'

And we also need to change what happens when this breakpoint's condition is met. As already mentioned, we want it to enable the other breakpoint. For that we check "**Enable Groups**" and enter the name of the group we used for the first breakpoint.

But, that is not enough. After all, this is a breakpoint and as soon as the condition is met, the application will get paused at it. Only we don't want that, we only want to pause at the other breakpoint. So we will uncheck the "**break**" property. This means the debugger will never pause here, it will only perform any other action that the breakpoint has.



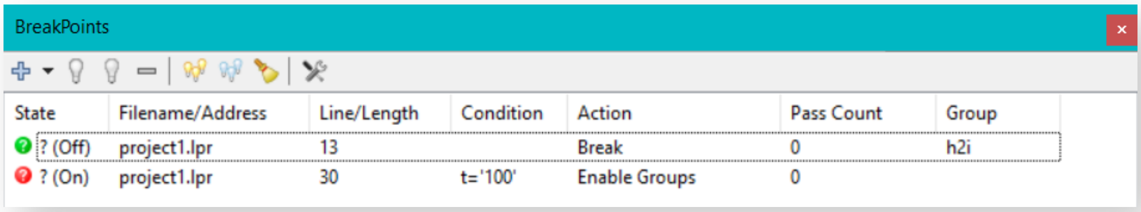
Actions:
☐ Break
☑ Enable Groups h2i
☐ Disable Groups

With this setup, we are ready to run the application. And again it will take us into "**Hex2Int**" and pause when the arguments will trigger the bug.

> **NOTE**: Instead of using a condition in the second breakpoint, we could have used the Hit-Count. However, there is a bug that prevents this combination from working.
> **So as of Lazarus 3.0 using Hit-Count and Enable-Groups together does not yet work.**

More breakpoint properties

As opportunity presents itself, lets take a look at the remaining properties for breakpoints. And also have a short mention of the breakpoint window, which can be opened from the menu **View → Debug Windows → Breakpoints** or using **Ctrl-Alt-B**


BreakPoints

State	Filename/Address	Line/Length	Condition	Action	Pass Count	Group
? (Off)	project1.lpr	13		Break	0	h2i
? (On)	project1.lpr	30	t='100'	Enable Groups	0	

It provides an overview of all your breakpoints, and shows some of the properties you have set. The image shows the 2 breakpoints from "**Let the caller decide**". You can see the first breakpoint has a group. And the second breakpoint has the "**Enable Groups**", but not the "**Break**" action.
The overview however does not show, which groups it will enable.

The window also provides tool-buttons to access the property dialog, as well as adding, removing, enabling and disabling breakpoints.

Hit Count and Pass Count

In the Breakpoint window we can see a column "**Pass Count**". It is a simple counter how often the breakpoint was hit. It only counts when the breakpoint is enabled. And depending on the backend, only if the break-condition is met.

This is the value against which the configured Hit-Count is compared. The breakpoint will act (*that is pause, or perform whatever other action is configured*) only if the **Hit-Count** is greater or equal than the **Pass-Count**.

With the **Pass-Count** you have a simple counter how often a line gets executed. Simply uncheck the "**break**" action, and watch the **Pass-Count**. Though, on very high frequented lines of code, even just counting may introduce a noticeable slow down.
As an example getting such a count may be useful for the first example. If we hadn't known how often the breakpoint needs to be passed before the error happened,
then we could have used the **Pass-Count**.

"Break" and Auto Continue

The "**Break**" action we already used in the third example. By default it is set, and the breakpoint will pause. If a breakpoint is set to perform other actions, then this can be unset. It can also be unchecked without any other action in place, making the breakpoint a simple counter.

Another effect of unchecking this is that the IDE will not take focus when such a breakpoint is hit. Together with "**Take Snapshot**" below, this can help debugging focus sensitive code.

"**Auto continue**" is similar, it will enter a pause state, but it wont transfer focus to the IDE.
It will wait the specified number of milliseconds and then continue with "**run**" (*as if* **F9** *was pressed*).

Since "**Auto continue**" uses "**run**", it will not work well, if you are stepping over a function, and hit an "**Auto continue**" breakpoint in this function. It will turn the "**Step Over**" into "**Run**". Unchecking "**Break**" will not interfere with stepping.

On the other hand "**Auto continue**" waits a moment. So you could look at **Watches**, **Locals** or other Windows that you have opened. And during the wait time you can also decide to hit the pause button, which will stop the counter and pause the app until you continue it yourself.

Enable and Disable Groups

Picking up from the earlier example, there are a few more details of interest.
In the example we placed the breakpoint with the "**Enable-Groups**" in the caller, but that isn't a must. Any breakpoint can enable or disable any other. So whenever your code does something that may affect other code, you can enable or disable breakpoints in the affected places.

Most often, this is either done as a one-off enable, or enabling, disabling breakpoints exist in pairs.
Another helpful option is, that a breakpoint can disable itself. This may be useful if you have a breakpoint that does log something or takes a snapshot. You might enable it from some other breakpoint, then it gets hit while enabled, and disables itself. So the logging only happens once after each time that the code passed through one of the breakpoints that will enable it.

You can have more than one breakpoint in each group, and a breakpoint can enable or disable several groups. And breakpoints that are in a group can at the same time enable or disable other breakpoints. So you can set up chains of enabling breakpoints.

Generally spoken, any breakpoint in a group is no different to any non-grouped breakpoints. Whenever it is enabled, it can have conditions, Hit-Count and all else.

Eval Expression, Log Message, Log CallStack
Those 3 actions will log data to the Event-log. "**Eval Expression**" will interpret its input as watch.
"**Log Message**" will log the literal text.
You can find the event log in the menu **View → Debug Windows → Event log** or open it by pressing **Ctrl-Alt-V.**
The event log shows a variety of messages about the debugged process. Which messages are shown can be decided in the options available via the context menu of the window. To see the log data from the breakpoints you need to enable "**Breakpoints**" in the checklist-box of the options.

Take Snapshot
This option will add a "**snapshot**" (*permanent*) entry into the debug history. If a breakpoints doesn't have this option, then when it pauses the application ("**break**" enabled), it will add a non-permanent entry. With this option the entry will be added to the list of permanent entries. And with this option, even a breakpoint that has "**break**" disabled will record a snapshot. This is an important feature, if you need to debug anything that is focus sensitive, or anything that would change behaviour if the IDE interrupted it for to long. (*Mind, when taking a snapshot, the application will still run slower, since the debugger needs some time to take the snapshot*).

A **snapshot** contains: The **thread** window, the stack window with the **top five stack-frames**, the **watches** and the **locals** window for the **top stack frame** of the current thread. It can contain more stack frames, if the stack window is open and set to display more frames. But it will still only take watches and locals for the top frame.

We could have debugged our initial issue using this feature. If we had set breakpoints in the function **`Hex2Int`** and collected snapshots without pausing, then once the application would have stopped with the wrong result, we could have looked through the snapshots and would have seen what happened.

DEBUG HISTORY

Whenever the debugger pauses the application it displays information such as the stack, watches and locals, and also a list of threads that the application has. The debug history keeps a copy of the data for the last 25 times the application paused.

Those entries are listed in the history window accessible from the menu **View → Debug Windows → History** or via **Ctrl-Alt-H.** Selecting an entry in the window will make the **Watches, Locals, Stack** and **Thread** window display the values from that entry instead of the current value.

If you need an entry to survive the 25 entry limit, you can press the "**Add Snapshot**" button. The entry will then be kept until you either delete it or you finished debugging and start a new debug session.
The breakpoint option "**Take snapsho**t" also adds such a permanent entry.

This allows to collect data and analyse it later. Some examples were this can be useful are:

- **Debugging of focus sensitive applications,** or apps that have a global mouse or keyboard hook and would behave differently if you would send key or mouse input to the IDE.
- **Collecting data until an error happens.** Then to go back and analyse what happened before the error.
- **Saving collected debug data** and send it to a co-worker for feedback.
  (*The debug history window has an import/export function*)
- **Having some one else running your app** in the debugger on their system. You would send them a project, with pre-configured breakpoints (*taking snapshots, but not* "**break**"**-ing**).
  And they would sent back the collected history.

Beginner Expert

BACKGROUND

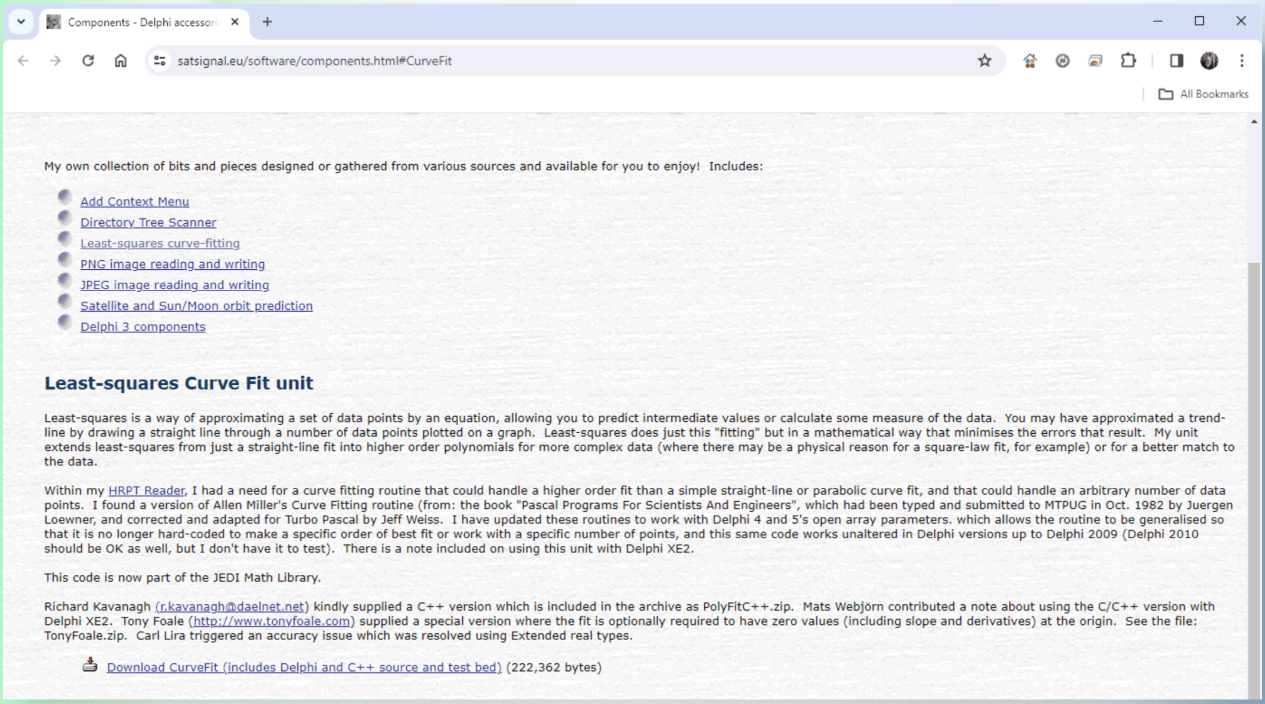
In the realm of precision metalworking, the ability to accurately model and reproduce intricate shapes and curves from a set of data points is crucial.
This challenge was presented to us when a client required software capable of driving a **metalworking machine** to produce smooth, accurate curves based on sampled points.
It had to produce an output which was a smooth curve, based on a set of sampled points.
Having worked on similar problems before, I had a rough idea of how this should work mathematically, but as is usually the case when developing software,
I was sure that someone else would have already solved this particular problem.

REQUIREMENTS

The primary goal was to develop software capable of generating a curve that closely fits a given set of points. This task involved not only finding the best-fit curve but also enhancing its resolution.
This was achieved by introducing additional points, ensuring a high degree of smoothness.
For instance, transforming an input of 400 points into an output comprising 5,000 points that lie precisely on the calculated curve.
This requirement is fundamental in applications where precision directly influences the final product's quality, such as in metalworking.

RESEARCH

A quick Google search identified some possible Delphi solutions to this problem.
The one that looked most promising was the least squares curve fit unit by **David Taylor:**
`https://www.satsignal.eu/software/components.html#CurveFit`
This was a single Delphi unit that did not depend on third-party components and simply used the `System.Math` unit.
The routine was written in Delphi 5, but the test project compiled in the latest version of Delphi without any issues, as one would expect with a purely mathematical library.
The main procedure is listed o the next page : Article Page 2/5


Components - Delphi accessori
satsignal.eu/software/components.html#CurveFit
All Bookmarks

My own collection of bits and pieces designed or gathered from various sources and available for you to enjoy! Includes:

- Add Context Menu
- Directory Tree Scanner
- Least-squares curve-fitting
- PNG image reading and writing
- JPEG image reading and writing
- Satellite and Sun/Moon orbit prediction
- Delphi 3 components

Least-squares Curve Fit unit

Least-squares is a way of approximating a set of data points by an equation, allowing you to predict intermediate values or calculate some measure of the data. You may have approximated a trend-line by drawing a straight line through a number of data points plotted on a graph. Least-squares does just this "fitting" but in a mathematical way that minimises the errors that result. My unit extends least-squares from just a straight-line fit into higher order polynomials for more complex data (where there may be a physical reason for a square-law fit, for example) or for a better match to the data.

Within my HRPT Reader, I had a need for a curve fitting routine that could handle a higher order fit than a simple straight-line or parabolic curve fit, and that could handle an arbitrary number of data points. I found a version of Allen Miller's Curve Fitting routine (from: the book "Pascal Programs For Scientists And Engineers", which had been typed and submitted to MTPUG in Oct. 1982 by Juergen Loewner, and corrected and adapted for Turbo Pascal by Jeff Weiss. I have updated these routines to work with Delphi 4 and 5's open array parameters. which allows the routine to be generalised so that it is no longer hard-coded to make a specific order of best fit or work with a specific number of points, and this same code works unaltered in Delphi versions up to Delphi 2009 (Delphi 2010 should be OK as well, but I don't have it to test). There is a note included on using this unit with Delphi XE2.

This code is now part of the JEDI Math Library.

Richard Kavanagh (r.kavanagh@daelnet.net) kindly supplied a C++ version which is included in the archive as PolyFitC++.zip. Mats Webjörn contributed a note about using the C/C++ version with Delphi XE2. Tony Foale (http://www.tonyfoale.com) supplied a special version where the fit is optionally required to have zero values (including slope and derivatives) at the origin. See the file: TonyFoale.zip. Carl Lira triggered an accuracy issue which was resolved using Extended real types.

Download CurveFit (includes Delphi and C++ source and test bed) (222,362 bytes)

```
procedure PolyFit (const x, y: array of real;
                   var coefs: array of real;
                   var correl_coef: real;
                   const npoints, nterms: Integer);
var
  error: boolean;
  i, j: integer;
  xi, yi, yc, srs, sum_y,sum_y2: real;
  xmatr: matrix;        // Data matrix
  a: matrix;
  g: array of real;     // Constant vector

begin
  if nterms < 1 then
    Raise EMathError.Create ('PolyFit called with less than one term');
  if npoints < 2 then
    Raise EMathError.Create ('PolyFit called with less than two points');

  SetLength (g, nterms);
  SetLength (a, nterms, nterms);
  SetLength (xmatr, npoints, nterms);

  for i := 0 to npoints - 1 do
    begin      { setup x matrix }
      xi    := x [i];
      xmatr [i, 0] := 1.0;               { first column }
      for j  := 1 to nterms - 1 do
      xmatr [i, j] := xmatr [i, j - 1] * xi;
    end;

  square (xmatr, y, a, g, npoints, nterms);
  GaussJordan (a, g, coefs, nterms, error);
  sum_y    := 0.0;
  sum_y2   := 0.0;
  srs      := 0.0;

  for i := 0 to npoints - 1 do
    begin
      yi := y [i];
      yc := 0.0;
      for j := 0 to nterms - 1 do
        yc     := yc + coefs [j] * xmatr [i, j];
      srs      := srs + sqr (yc - yi);
      sum_y    := sum_y + yi;
      sum_y2   := sum_y2 + yi * yi
    end;

  // If all Y values are the same, avoid dividing by zero
  correl_coef := sum_y2 - sqr (sum_y) / npoints;
  // Either return 0 or the correct value of correlation coefficient
  if correl_coef <> 0 then correl_coef := srs / correl_coef;
  if correl_coef >= 1
    then correl_coef := 0.0
    else correl_coef := sqrt (1.0 - correl_coef);

  g := nil;
  a := nil;
  xmatr := nil;
end;
```

USING THE LIBRARY

The core function of the selected library required inputs in the form of arrays of X and Y values, along with a specification of the number of terms (*degree of the polynomial*) to use.
This aspect is critical as it determines the complexity and accuracy of the resulting curve.
A single term would generate a simple horizontal line, while increasing the terms introduces higher degrees of **polynomials**, offering a more precise fit at the cost of computational complexity.

INPUTS

The main function accepts:

- An array of floating-point X values.
- An array of floating-point Y values.
- The number of terms to use.

The number of terms relates to the degree of the polynomial formula which will be used.
A single term equates to a horizontal line on a graph, e.g. $y = 2$
The use of two terms equates to a straight line on a graph, e.g. $y = 3x + 2$
Using three terms equates to a quadratic equation, e.g. $y = 4x^2 + 3x + 2$

OUTPUTS

The calculation function will populate an array of floating-point values which correspond to the coefficients of the equation. For example, if we want to use three terms (*a quadratic equation*), we can pass an array of three `floating-point` numbers. In the quadratic example above, the coefficients would be 2, 3, and 4, respectively.
The calculation also returns a correlation coefficient, which tells us how closely the curve correlates with the input points provided. The closer this coefficient is to 1, the more accurate the fit.

RESULTS

When using the library, unit tests were written using a set of test points provided by our customer. The points provided were typical of the samples they would be using in production.

With these sample points, the most accurate curve was produced when calling the function with six terms. This equates to a **polynomial** function of the fifth degree.

When using more than six terms, occasional floating-point overflow errors were produced, presumably because the calculated coefficients were too small to be represented in a Double type in **Delphi**.

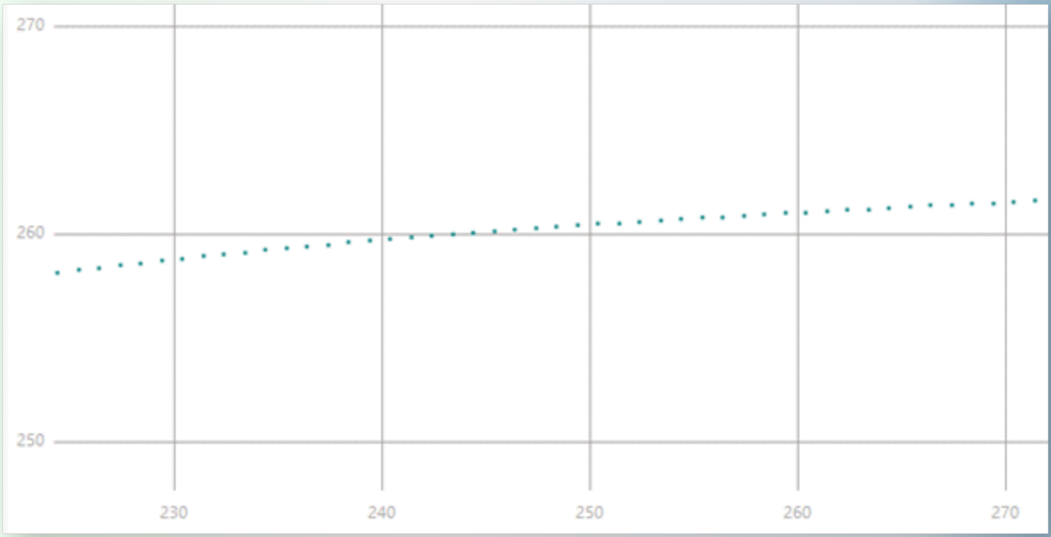
When using fewer than six terms, the routine still calculated a curve of best fit, but of course, the fit was not as accurate. With fewer terms, the correlation coefficient was lower.

With six terms, we received six coefficients back in the array provided to the function.

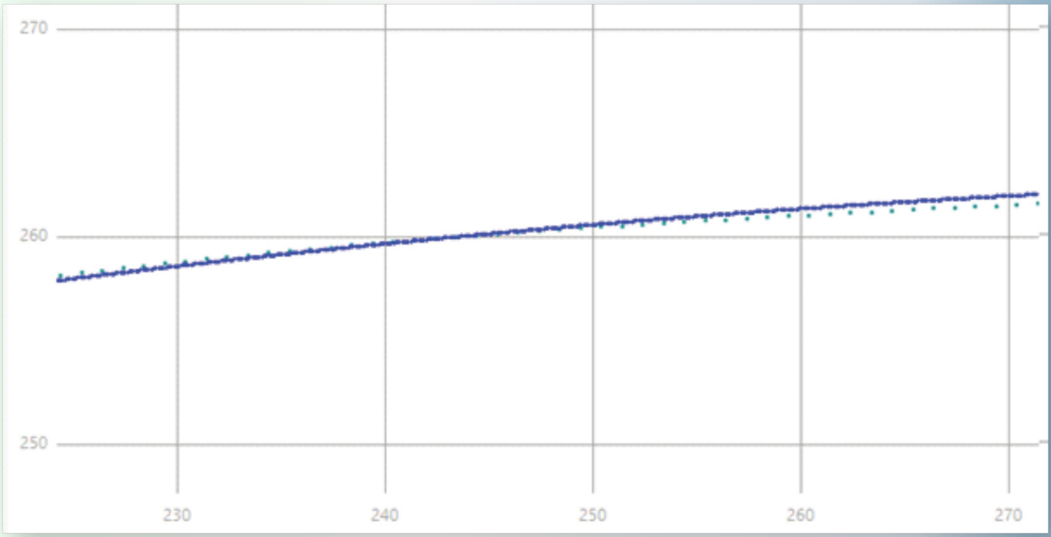
Once we have the coefficients, we can calculate the Y value for any value of X.
To achieve the desired curve, an array of 5,000 evenly spaced X points was calculated, bounded by the maximum and minimum X values of the sample.

```
function CalculateY(const X: Double; const Coefficients: TArray<Double>): Double;
begin
  var yc := 0.0;
  var xc := 1.0;

  for var i := Low(Coefficients) to High(Coefficients) do
  begin
    yc := yc + Coefficients[i] * xc;
    xc := xc * X;
  end;

  Result := yc;
end;
```

Here is a sample of some of the input points:



In the software, the resulting best-fit curve is calculated and overlaid over the points as follows:

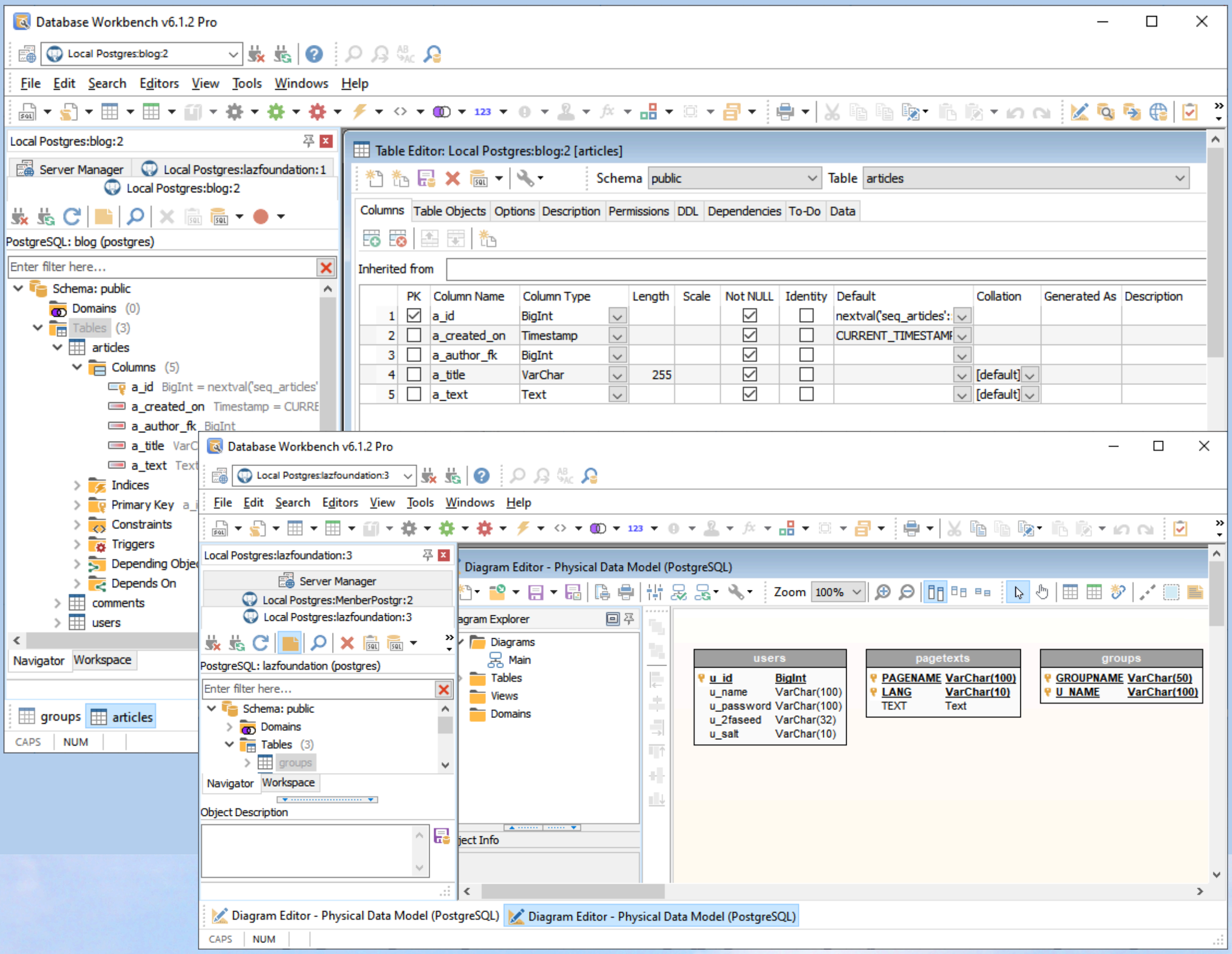


IMPLEMENTATION

The function uses a least-squares curve fitting algorithm using **Gauss-Jordan elimination**.
This is a well-known algorithm for finding a best-fit polynomial curve for a set of points.

CONCLUSION

We have seen that a mathematical library originally written in **Turbo Pasca**l and early versions of **Delphi** can be used in the latest versions of **Delphi** without modification.
This library can produce accurate and smooth curves for the best fit of a set of data points.
The higher the number of terms and hence the degree of the polynomial, the more accurate the fit.



Schema: public Table: articles

	PK	Column Name	Column Type	Length	Scale	Not NULL	Identity	Default	Collation	Generated As	Description
1	✓	a_id	BigInt			✓		nextval('seq_articles':			
2		a_created_on	Timestamp			✓		CURRENT_TIMESTAMF			
3		a_author_fk	BigInt			✓					
4		a_title	VarChar	255		✓			[default]		
5		a_text	Text			✓			[default]		

Zoom 100%

users	
u_id	BigInt
u_name	VarChar(100)
u_password	VarChar(100)
u_2faseed	VarChar(32)
u_salt	VarChar(10)

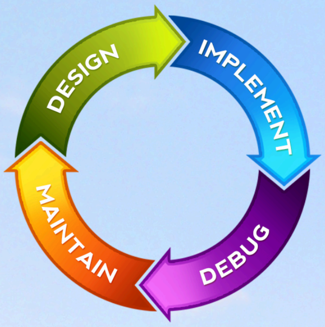
pagetexts	
PAGENAME	VarChar(100)
LANG	VarChar(10)
TEXT	Text

groups	
GROUPNAME	VarChar(50)
U_NAME	VarChar(100)

Introducing

Database Workbench 6

database development environment



Consistent user interface, modern code editors, Unicode enabled, HighDPI aware, ER designer, reverse engineering, meta data browsing, visual object editors, meta data migration, meta data compare, stored routine debugging, SQL plan visualizer, test data generator, meta data printing, data import and export, data pump, Grant Manager, DBA tasks, code snippets, SQL Insight, built in VCS, report editor, database meta data search, numerous productivity tools and much more...

for SQL Server, Oracle, MySQL, MariaDB, Firebird, InterBase, NexusDB and PostgreSQL

Upscene

Database tools for developers

www.upscene.com

PUSH NOTIFICATION SUPPORT IN FREE PASCAL

BY MICHAEL VAN CANNEYT

Starter Expert

ABSTRACT
Sending a message to a smartphone or a web application is part of many applications.
Usually these messages are sent by a background service. **Free Pascal** contains units with which you can send **Push notifications**. A closer look.

❶ INTRODUCTION

There are times when you may wish to send a message to people in your community, or your users, to notify them of an interesting or relevant event or a piece of news.

For the browser, there is a standard called '**WebPush**' which allows you to send messages to a browser.
The browser keeps a background process (*so-called service workers*) running which listens for such messages, and displays them when they arrive.

Smartphones have their own version of the messaging protocol:
depending on the OS (**Android or iOS**), the **API** is different.

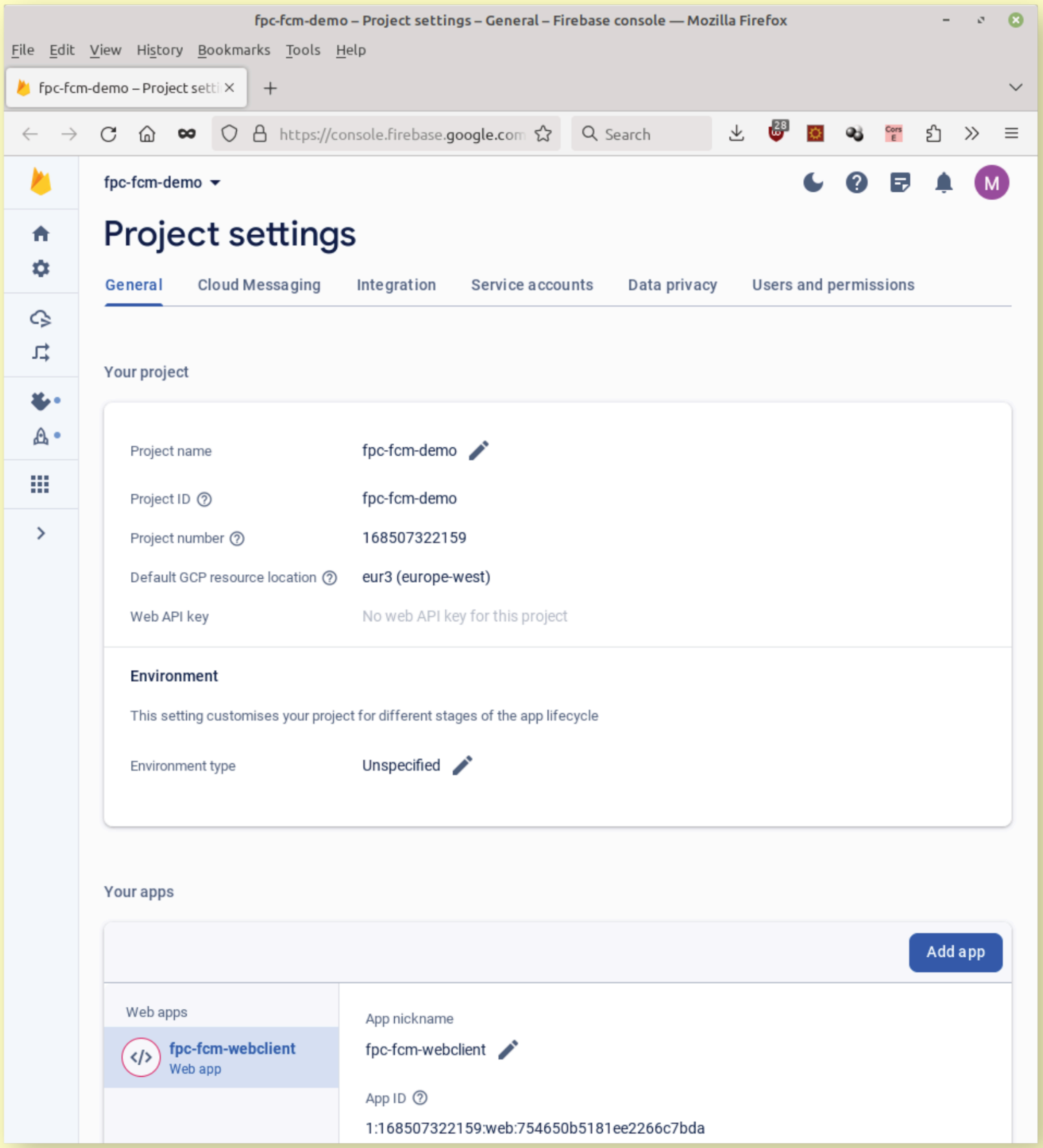
Regardless of the platform, the mode of operation is the same:
The process starts by asking the user permission to show him or her notifications.
Once the permission is granted, the OS or the browser can subscribe to the **push message service,** which returns a unique token that can be used to send messages to the device on which the user granted permission. The token is valid till the user retracts his permission.

Google offers a service which allows you to use such a token to send a message, regardless of the platform which issued the token: **Firebase Cloud Messaging (FCM).**
It allows you to send messages, and provides you with statistics on how the users reacted on your messages.
This service can be accessed using a simple REST API.
Thanks to some generous sponsoring, **Free Pascal** contains a unit which allows you to use this API without the need to worry about the details of the API.
In what follows, we'll examine this unit.

❷ PREREQUISITES

Before we dive into the API, some preparations must be made. FCM is not a free service, and if you plan to send a lot of messages, you will need to pay.
It should not come as a surprise that Google requires you to register the application that you wish to use to send messages.
If you have not done this before, then you need to start by creating a **Firebase project** in the **Google Firebase console**: (*See page 2.of this article*)
`https://console.firebase.google.com/`

❷ PREREQUISITES


fpc-fcm-demo – Project settings – General – Firebase console — Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://console.firebase.google.com
Search
fpc-fcm-demo
General
Cloud Messaging
Integration
Service accounts
Data privacy
Users and permissions
Add app

Project settings

Your project

Project name	fpc-fcm-demo
Project ID	fpc-fcm-demo
Project number	168507322159
Default GCP resource location	eur3 (europe-west)
Web API key	No web API key for this project

Environment

This setting customises your project for different stages of the app lifecycle

Environment type	Unspecified

Your apps

Web apps
fpc-fcm-webclient
Web app

App nickname
fpc-fcm-webclient

App ID
1:168507322159:web:754650b5181ee2266c7bda

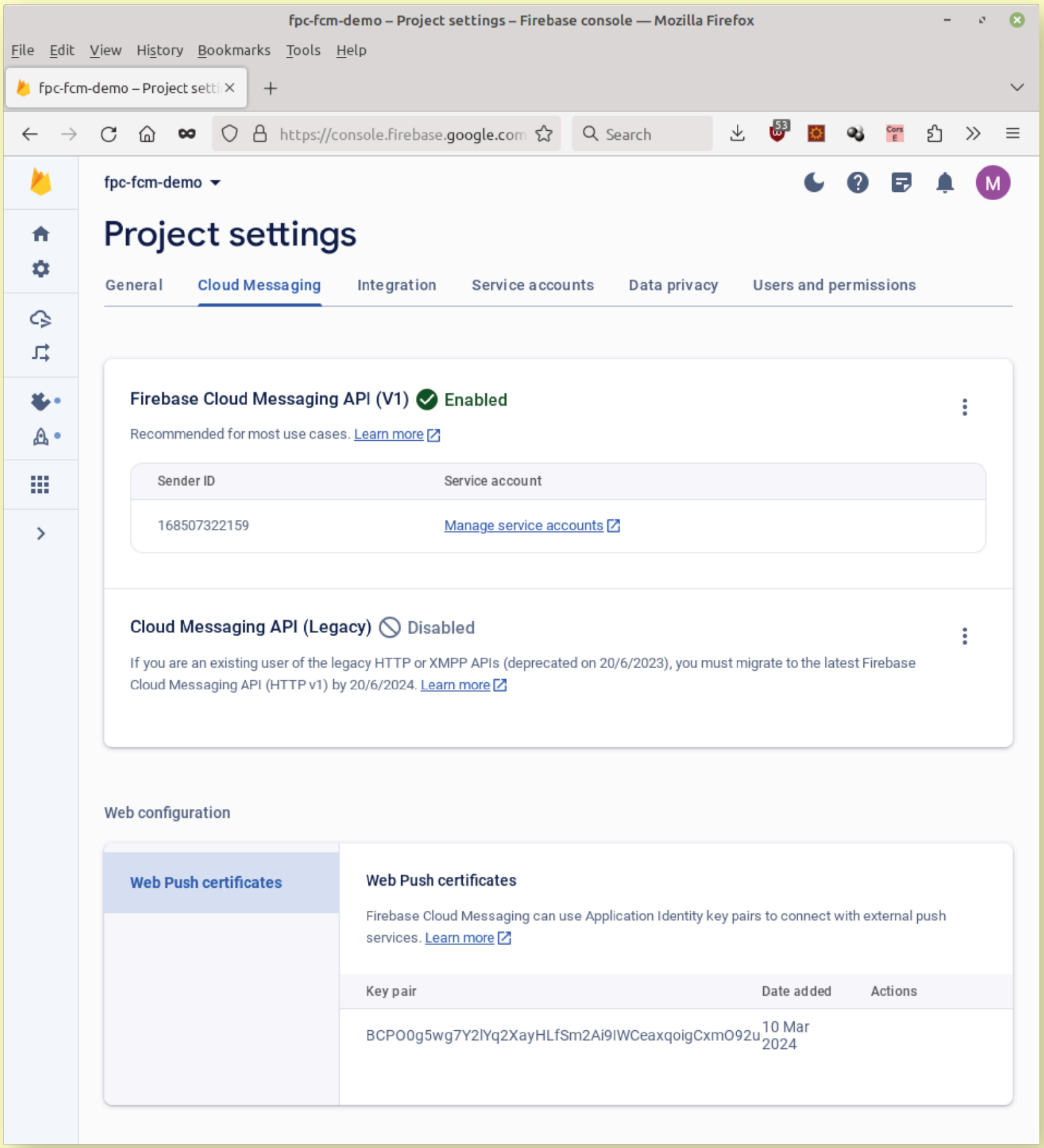
Figure 1: A finished firebase project

You will of course need to have a google account to log in to this service. **Any valid Google account can be used**. The project dropdown contains a menu item to create a new project.
A complete walk through of the creation of a project is outside the scope of this article,
but the process is simple and well-documented elsewhere (*although the details of the screens change regularly*).

When done, you should end up with something like figure *1 on page 2 of this article.*

The next step is to **register** the application that will access the **Google Firebase service**.
Multiple applications **can use the same Firebase Cloud Messaging project,** and you should register them separately:
On the page shown in *figure 1 on page 2 of this article* you can see a button with which you can create a new application.

❷ PREREQUISITES


fpc-fcm-demo – Project settings – Firebase console — Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://console.firebase.google.com
Search
fpc-fcm-demo

Project settings

General | Cloud Messaging | Integration | Service accounts | Data privacy | Users and permissions

Firebase Cloud Messaging API (V1) Enabled

Recommended for most use cases. Learn more

Sender ID	Service account
168507322159	Manage service accounts

Cloud Messaging API (Legacy) Disabled

If you are an existing user of the legacy HTTP or XMPP APIs (deprecated on 20/6/2023), you must migrate to the latest Firebase Cloud Messaging API (HTTP v1) by 20/6/2024. Learn more

Web configuration

Web Push certificates

Web Push certificates

Firebase Cloud Messaging can use Application Identity key pairs to connect with external push services. Learn more

Key pair	Date added	Actions
BCPO0g5wg7Y2lYq2XayHLfSm2Ai9IWCeaxqoigCxmO92u	10 Mar 2024	

Figure 2: Cloud Messaging page with VAPID Key

The application definition is important: when the application is created, you can download a JSON application configuration file which you need to use when **accessing the Google FCM APIs**: among other things it contains a unique project identifier.
The file contents will look something like this:

```
{ apiKey: "XYZ",
  authDomain: "fpc-fcm-demo.firebaseapp.com",
  projectId: "fpc-fcm-demo",
  storageBucket: "fpc-fcm-demo.appspot.com",
  messagingSenderId: "123",
  appId: "1:123" }
```

❷ PREREQUISITES

If you wish to send messages to users in the browser, you will need a **VAPID** key.
This is a special **key identifying your browser application**, and is required by the **WebPush protocol** that is used by **Firebase** to send messages to the browser.

When your **Firebase** project is created, you can get a **VAPID key** from the project page on the **'Cloud Messaging'** page (*see figure 2 on page 3*), below **'Web Push Certificates'**.

The public key needs to be copied, as it will have to be used in the browser.

The last step is to create a **Service Account:**
The service account is needed to actually send messages. This account is used to get an **access token** for **authenticating** the requests to the **Firebase HTTP** send **REST APIs**. You can get the **service account private key** on the **'Service accounts'** page from your project, shown in *figure 3 on page 5 of this article.*

Just like the application info **configuration** file, you need to download the **service account info** file, and save it somewhere on your harddisk. It should look something like the following:

```
{ "type": "service_account",
   "project_id"     : "fpc-fcm-demo",
   "private_key_id": "123456789",
   "private_key"   : "XXXYYYZZZ",
   "client_email"  : "firebase-adminsdk@gserviceaccount.com",
   "client_id"     : "987654321",
   "auth_uri"      : "https://accounts.google.com/o/oauth2/auth",
   "token_uri"     : "https://oauth2.googleapis.com/token",
   "auth_provider_x509_cert_url":
      "https://www.googleapis.com/oauth2/v1/certs",
   "client_x509_cert_url": "https://www.googleapis.com/",
   "universe_domain": "googleapis.com" }
```

When all these steps are completed, you have all information to get started.

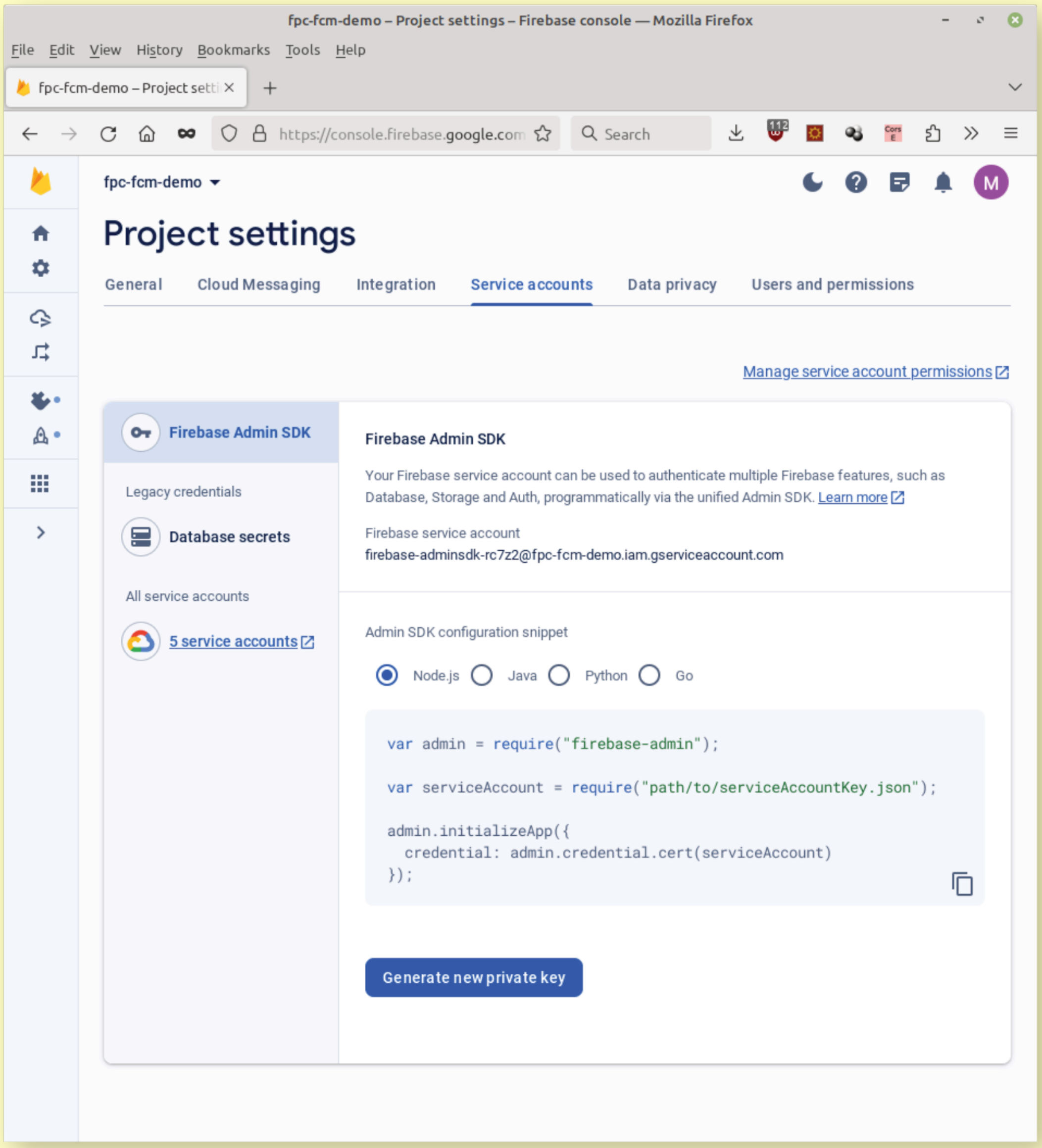
❸ THE MESSAGE OBJECT

To send a message to the **Firebase Cloud Messaging APIs** means to send a **JSON** object.
The following page describes the message that can be sent:
https://firebase.google.com/docs/reference/fcm/rest/v1/projects.messages#resource:-message

The `fpfcmtypes` unit contains an **Object Pascal** class definition that offers you all the fields present in the specification. This has the advantage that **you can use code completion**.

```
TNotificationMessage = class(TJSONPersist)
public
  procedure ToJSON(aObj : TJSONObject);
  function Encode : string;
  procedure Clear;
  // toplevel properties, valid for all platforms.
  // Notification data.
  Property Recipient : String;
  Property RecipientType : TRecipientType;
  property Data: TStrings;
  property Title: string;
  property Body: string;
  property Image: string;
  // available in Apple and Android, not in web.
  Property Options: TMessageOptions;
  Property SendOptions : TNotificationSendOptions;
  // Apple specific
  Property AppleConfig : TAppleConfig;
// Android specific
  Property AndroidConfig : TAndroidConfig;
// Web specific
  Property WebPushConfig : TWebPushConfig;
// FCM options
  Property FCMOptions : TFCMOptions;
end;
```

❸ THE MESSAGE OBJECT



Firebase Admin SDK

Your Firebase service account can be used to authenticate multiple Firebase features, such as Database, Storage and Auth, programmatically via the unified Admin SDK. Learn more

Firebase service account
firebase-adminsdk-rc7z2@fpc-fcm-demo.iam.gserviceaccount.com

Admin SDK configuration snippet

[✓ Node.js] Java Python Go

```
var admin = require("firebase-admin");

var serviceAccount = require("path/to/serviceAccountKey.json");

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});
```

Figure 3: The service account info page

The meaning of the **Recipient, Title , body and Image** properties should be intuitively clear.
The `RecipientType` property is of type `TRecipientType` and can be one of `rtToken, rtTopic, rtCondition`:
you can **send messages to one person**, or **multiple persons**.

The **`AppleConfig`**, **`AndroidConfig`**, **`WebPushConfig`** and **FCMOptions** contain specific configuration options which **FCM** will use when sending the message to one of these platforms.

For day-to-day use you will probably not need to set these, but they are all detailed in the webpage above:
the properties of these objects reflect the properties specified in the specification.

❸ THE MESSAGE OBJECT

The `SendOptions` property is a set which tells the message object which of these optional configuration parts it should include in the **JSON** message.

Once you have filled the object properties, the `Encode` function will return a string that contains the **JSON** message to send to the **FCM** server. The `Clear` method will clear all properties.

So, to send the same message to multiple users, you fill the message properties, and in a loop set the recipient, retrieve the **JSON** and send the **JSON** to the **FCM** server.

❹ THE CLIENT OBJECT

The `fpfcmclient` unit contains the `TFCMClient` object. This class is the client used to communicate with the FCM server. It takes care of authentication and sending the message. The class does not have a lot of methods or properties:

```
TFCMClient = class(TComponent)
Public
   Procedure InitServiceAccount(const aFileName: string;
                                               aRoot: TJSONStringType);
   Procedure InitServiceAccount(const aJSON : TJSONObject);
   function Send(aMsg : TNotificationMessage; aRecipient : UTF8String) : Boolean;
   function Send(aMsg : TNotificationMessage; aRecipients :
                                               Array of UTF8String) : Boolean;
   Property WebClient : TAbstractWebClient;
   Property BearerToken: TBearerToken;
   Property ServiceAccount: TServiceAccountData;
   Property LogFile: String;
   Property OnError: TFCMErrorEvent;
   Property OnNewBearerToken : TFCMBearerTokenEvent;
   Property OnResponse: TFCMResponseEvent;
end;
```

The properties are quite simple:

WebClient
This can be set to an instance of an abstract `TWebClient` class.
This class abstracts away the details of the HTTP protocol, we'll show how to use this.
If you don't set it, then the `TFCMClient` class will create a default instance.

BearerToken
The bearer Authentication token to be used in the next HTTP request.
You can set this if you stored it somewhere. If none was set or the token is expired, a bearer token will be fetched as needed using the service account details.

ServiceAccount
This property provides read-only access to the Service account data.
This property must be initialized with the **`InitServiceAccount`** method.

LogFile
Set this property to the name of a file if you want a log of the HTTP communications with google FCM servers: all headers and bodies of request and response will be written to this file.

The following events are fully optional:

`OnError`
You can set this event if you wish to handle errors yourself. If not set, an exception is raised on error.

`OnNewBearerToken`
as mentioned before, a bearer token will be fetched as needed using the service account details. This event is called when a new access token was received. You can use it to store it for the next call.

`OnResponse`
this event is called with the send response: you can store the response in a database if you so desire.

❹ THE CLIENT OBJECT

The class has only 2 public methods:
`InitServiceAccount` You can call this with the name of a file and an optional path to a root element. This will initialize the `ServiceAccount` data. The file to provide is the service account configuration file you downloaded when you set up the service account. You can also obtain the **JSON** data by some other means and directly provide the **JSON** data.
Send This will send the message to the the **FCM** server using the **HTTP REST Api.** You can specify one or multiple recipients. The `Recipient` field of the message will be filled with each of the specified recipients, and the message will be sent. The function returns `True` if the message was sent successfully (*if you don't handle errors using OnError, an exception is raised when something goes wrong.*)

Armed with this class, how can we send a push notification ? Quite simply.
To demonstrate this, We create a console application based on `TCustomApplication`. In its `DoRun` method, we enter the following code:

```
Recip  :=GetOptionValue('r','recipient');
Msg    :=nil;
Client :=TFCMClient.Create(Self);
Try
  ConfigureClient(Client);
  Msg  :=TNotificationMessage.Create;
  ConfigureMessage(Msg);
  Client.Send(Msg,Recip);
Finally
  Msg.Free;
  Client.Free;
end;
```

The most work happens in the **ConfigureClient** and **ConfigureMessage** calls:
To configure the client, we need several items, all of which will be provided through the command-line options of our program. The first thing is to initialize the service account.
The file with the **service account information** can be specified using the **-s** command-line option, but a default filename is used if it was not specified:

```
procedure TFCMApplication.ConfigureClient(aClient : TFCMClient);

const Err = 'No service account configuration file found: %s';

var CfgFile : string;

begin
  // Service account info
  CfgFile:=GetOptionValue('s','service-account');
  if CfgFile='' then
  CfgFile:=ChangeFileExt(ParamStr(0),'-service-account.json');
  if not FileExists(CfgFile) then
  Raise EInOutError.CreateFmt(Err,[CfgFile]);
  aClient.InitServiceAccount(CfgFile,'');
  // Access token reuse
  if HasOption('a','access-token') then
  begin
    FAccessTokenFile:=GetOptionValue('a','access-token');
    // Load initial token
    if FileExists(FAccessTokenFile) then
      aClient.BearerToken.LoadFromFile(FAccessTokenFile);
    // Set handler so we save the token when it was fetched.
    aClient.OnNewBearerToken:=@DoHandleNewToken;
    end;
  // Log file
  if HasOption('l','log') then
    aClient.LogFile:=GetOptionValue('l','log');
end;
```

❹ THE CLIENT OBJECT

Using the **-a** option you can specify the bearer token:
the bearer token record has a method to load the token from file, or to save it to file. you can use these methods to reuse the same token. Normally a token has a life time of about 1 hour, after which a new token must be fetched. Lastly the log file can be set using the **-l** option.
The `DoHandleNewToken` method is called when a new token is requested. In this method, the token can be saved to file:

```
procedure TFCMApplication.DoHandleNewToken(Sender: TObject;
                                           const aToken: TBearerToken);
begin
  aToken.SaveToFile(FAccessTokenFile);
end;
```

The `ConfigureMessage` method sets the properties of the `notification` message.
You can specify a **JSON** file to load the message from (*using the* **-m** *option*).
Or you can specify the message body, title and image with the **-b, -t -i** options, respectively.

```
procedure TFCMApplication.ConfigureMessage(Msg : TNotificationMessage);
begin
  if HasOption('m','message') then
    LoadMessageFromFile(Msg,GetOptionValue('m','message'));
  if HasOption('t','title') then
    Msg.Title :=GetoptionValue('t','title');
  if HasOption('b','body') then
    Msg.Body  :=GetoptionValue('b','body');
  if HasOption('i','image') then
    Msg.Body  :=GetoptionValue('i','image');
end;
```

The `LoadMessageFromFile` is again quite simple:

```
procedure TFCMApplication.LoadMessageFromFile(Msg : TNotificationMessage;
    const aFileName : string);
Var
  F   : TFileStream;
  D   : TJSONData;
  Obj : TJSONObject absolute D;

begin
  D:=Nil;
  F:=TFileStream.Create(aFileName,fmOpenRead or fmShareDenyWrite);
  try
    D:=GetJSON(F);
    if not (D is TJSONObject) then
      Raise EFCM.CreateFmt('Invalid JSON data in message file %s',[aFileName]);
    Msg.Title :=Obj.Get('title',Msg.Title);
    Msg.Body  :=Obj.Get('body',Msg.Body);
    Msg.Image :=Obj.Get('image',Msg.Image);
  finally
    D.Free;
    F.Free;
  end;
end;
```

As you can see, the message file is a simple **JSON** file with 3 keys: title, body and `image`.
With that, the client is almost ready to be used. There is one small detail to take care of:
the **WebClient** property is not set anywhere. The `TFCMClient` class will then create a default webclient. The default **webclient** needs to be configured, and this can be done by adding the following units to the uses clause:

❹ THE CLIENT OBJECT **fphttpwebclient**

this sets the default web client to a class based on TFPHTTPClient;

opensslsockets

this enables support for HTTPS for TFPHTTPClient. HTTPS is needed to communicate with the FCM services.

Additionally, in the program start code, we need to add the following line:

```
DefaultWebClientClass:=TFPHTTPWebClient;
```

This will instruct the `TFCMClient` class to use the `TFPHTTPWebClient` class when creating a `TWebClient` instance.

With this code in place
(*plus some helper code to show a usage message and some checks for the command-line options*), the program can be executed from the command-line as follows:

```
sendmsg -m message.json -s service.json -a token.json -r TOKEN
```

Here TOKEN needs to be replaced with a token obtained by asking the user for permission to send him (or her) a message. The service account data will be loaded from file service.json (which you should have downloaded using the FCM project console).

The message is specified in the file `message.json`:

```
{
"title" : "A nice message",
"body" : "With a nice body",
"image" : "https://www.freepascal.org/favicon.png"
}
```

If you specify the command several times with different recipient tokens, you'll see that the token is saved in the `token.json` file and reused for the next calls.

❺ OBTAINING A TOKEN USING A WEBSITE

In the above, we have not shown how to get a token for sending a message to a user, we assumed it was available. In practice, this token must be obtained from the user by asking his permission to send him messages. This is a function that is available in mobile operating systems and in the browser.

We'll use **PAS2JS** to demonstrate how to get such a token and use it to send a message to the browser. The browser has a simple interface to show notifications, which is - unsurprisingly - called **Notification** and which is detailed here:

`https://developer.mozilla.org/en-US/docs/Web/API/notification`

The method to request permission to show notifications is
- can it be more simple? - called `requestPermission.`

When called, the browser will pop up a message asking for your permission to show you your notifications. The return value is a **Promise**, which will be fulfilled when the user has granted (*or denied*) the permission. After obtaining permission, you can get the token that you can use to send messages.
To do this, the Push manager can be used:

`https://developer.mozilla.org/en-US/docs/Web/API/PushManager`

The `subscribe` call will result in a subscription, which can be converted to a token.
The developers of **Firebase** have created a little **API layer** around this call, and we will follow their guidelines to get and use a token. This is a simple precaution: in fact it is not clear from the **Firebase** documentation whether **Firebase** simply uses the raw token obtained from the browser or adds some information to it which it needs to deliver the message.

The **Firebase Cloud Messaging API** has been made available in the `firebaseapp` unit.
We will use it to get a token, and we'll send this token to a small **HTTP server application**, which will use it to send a Push notification using the `TFCMClient` component presented above.

❺ OBTAINING A TOKEN USING A WEBSITE

So, to this end, we create a new "**Web Browser Application**" project in Lazarus.
This will create a HTML page and a program file. In the HTML page, some files need to be added to be able to use the **firebase API:**

```
<script src="firebase-app-compat.js"></script>
<script src="firebase-messaging-compat.js"></script>
```

These files can be downloaded from the firebase site, as described under:

`https://firebase.google.com/docs/web/alt-setup`

We're using the compatibility layer (*because that is what is described in the* `firebaseapp` *unit*), but it would also be possible to use the **modular API**.

To initialize a **Firebase application**, the application configuration file which you created and downloaded when you defined your project in the Firebase console needs to be used.
You can include the **JSON object definition** in the application code, but you can also put it in a file on your HTTP server:

```
var firebaseConfig = {
authDomain: "fpc-fcm-demo.firebaseapp.com",
projectId: "fpc-fcm-demo",
storageBucket: "fpc-fcm-demo.appspot.com",
messagingSenderId: "123",
appId: "1:123"
};
```

You can then include this file (*we'll name it* `config.js`) in the main HTML File of your project together with the include of your project file:

```
<script src="config.js"></script>
<script src="webclient.js"></script>
```

The rest of the HTML is quite simple:
we add 1 edit control (`edtMessage`) and 2 buttons (`btnSend` and `btnRegister`) to the HTML, plus some **`DIV`** tags to display the token and some output of the program.

```
<div class="container">
  <div class="box">
    <h3 class="title is-3">FCM Push notification demo</h3>
    <div class="field">
      <label class="label">Message to send:</label>
    <div class="control">
      <input id="edtMessage" class="input" type="text" placeholder="Message to send">
    </div>
  </div> <!-- .field -->

  <div class="field is-grouped">
    <div class="control">
      <button id="btnSend" class="button is-link" disabled>Send</button>
    </div>
      <div class="control">
        <button id="btnRegister" class="button is-link is-light">Register</button>
      </div>
    </div> <!-- .field -->
  </div> <!-- .box -->
  <div id="pnlToken" class="box is-hidden">
    <p>Your token: <span id="lblToken">?</span> <p>
  </div> <!-- .box -->
</div> <!-- .container -->
<script>
rtl.run();
</script>
<div id="pasjsconsole"></div>
```

The pasjsconsole element is where `WriteLn` feedback will be displayed.
The application code is in a `TDemoApp` class, a descendent of the TBrowserApplication class that comes standard with Pas2JS. When the application starts, the DoRun method is called. In it, the following code is executed to initialize some fields representing the various div HTML elements and the buttons. For the latter it also sets the click callbacks:

❺ OBTAINING A TOKEN USING A WEBSITE

```
var config : TJSObject; external name 'firebaseConfig';
procedure TDemoApp.DoRun;
begin
  RPCModule   :=TRPCModule.Create(Self);
  pnlToken    :=GetHTMLElement('pnlToken');
  lblToken    :=GetHTMLElement('lblToken');
  edtMessage  :=TJSHTMLInputElement(GetHTMLElement('edtMessage'));
  btnSend     :=TJSHTMLButtonElement(GetHTMLElement('btnSend'));
  btnSend.addEventListener('click',@HandleSend);
  btnRegister :=TJSHTMLButtonElement(GetHTMLElement('btnRegister'));
  btnRegister.addEventListener('click',@HandleRegister);
  Writeln('Initializing application...');
  App:=Firebase.initializeApp(config);
  App.messaging.onMessage(@HandleReceivedMessage);
  RegisterServiceWorker;
end;
```

The last lines initialize the **Firebase API** using the config object from our `config.js` file, and sets the `OnMessage` event handler of the **firebird** messaging API. The `OnMessage` event can be used to react to messages. Here we'll just display the notification message by creating a **TJSNotification** instance:

```
procedure TDemoApp.HandleReceivedMessage(aMessage: TJSObject);
var
  Notif: TJSObject;
  Opts : TJSNotificationOptions;
begin
  if assigned(aMessage) then
    console.debug('Message received: ',aMessage);
  Notif      :=TJSObject(aMessage['notification']);
  Opts       :=TJSNotificationOptions.new;
  Opts.body  :=string(Notif['body']);
  Opts.image :=string(Notif['image']);
  TJSNotification.new(string(Notif['title']),opts);
end;
```

We can do this because we requested permission from the user to display notifications.
Note that when the webpage is not loaded and focused, the service worker will display the message (*also using the Notification API of the browser*).
The last step when initializing the application is to register a service worker script:

```
procedure TDemoApp.RegisterServiceWorker;
begin
 Window.Navigator.serviceWorker.register('firebase-messaging-sw.js').
  _then(function (js : JSValue) :JSValue
   begin
     reg:=weborworker.TJSServiceWorkerRegistration(js);
     if assigned(Reg) then
       Writeln('Registered service worker...')
   end,function (js : JSValue) :JSValue
   begin
     Writeln('Unable to register service worker')
   end);
end;
```

In the above code, the service worker script is called "`firebase-messaging-sw.js`", it can be downloaded from the **Firebase** documentation pages. The `register` method returns a **promise**, which resolves to a **service worker registration object.** Here we just save the registration in the `reg` field for later use, and display a message to show whether the service worker was successfully registered or not.

For more elaborate web pages, more things can be done once the **service worker** is registered, such as establishing a message channel between the service worker and the main web page.
For our current example, this is not needed. A service worker is a small service, maintained by the browser: It will run in the background, and one thing it can do is to listen for incoming messages - exactly what it needs to do for our demo.

The service worker script does not do much, except initializing the firebase messaging application:

❺ OBTAINING A TOKEN USING A WEBSITE

```
importScripts('https://www.gstatic.com/firebasejs/9.2.0/firebase-app-compat.js');
importScripts('https://www.gstatic.com/firebasejs/9.2.0/firebase-messaging-compat.js');
importScripts('config.js');
firebase.initializeApp(firebaseConfig);
```

With all this, the browser application is ready to receive push messages.
To get a **Firebase messaging** token, the user must click the '**Register**' button. In the 'click' handler of the register button (**HandleRegister**), the service worker registration which we saved earlier is passed on to the `getToken` call of the Firebase messaging API to get a Firebase messaging token.

```
TFirebaseMessaging = class external name 'firebase.messaging.Messaging' (TJSObject)
function getToken (options : TMessagingGetTokenOptions): string; async;
end;
```

The `TMessagingGetTokenOptions` object has a serviceworkerRegistration field:
When set, the firebase API knows which service worker will be handling the receipt of messages.
Additionally, the `vapidkey` field must be set to the **VAPID key** you created when the Firebase project was created. The **VAPIDKey constant** (*not shown here*) contains this key.

```
procedure TDemoApp.HandleRegister(event: TJSEvent);
var
  Token : string;
  opt : TMessagingGetTokenOptions;

begin
  opt:=TMessagingGetTokenOptions.New;
  opt.serviceworkerRegistration:=self.Reg;
  opt.vapidKey:=TheVAPIDKey;
  Token:=Await(App.messaging.getToken(opt));
  if (token='') then
    RequestPermission
  else
    HaveToken(token);
end;
```

The `GetToken` call is an **Async** call, so it returns a promise. Using the **Await** builtin, we can transform it to an actual token string. If the token string is empty, the firebase API does not yet have a token, and we must request the user his or her permission to show notifications. If a non-empty token is returned, we can send it to the server. This is done using the `RequestPermission` and `HaveToken` calls, respectively:

```
procedure TDemoApp.requestPermission;
  function onpermission (permission : jsvalue) : jsvalue;
  var
  token : string;
  begin
    if (permission='granted') then
      begin
        writeln('Notification permission granted.');
        handleregister(nil);
      end;
    end;
begin
  Writeln('Requesting permission...');
  TJSNotification.requestPermission()._then(@OnPermission)
end;
```

The `RequestPermission` class method of `TJSNotification` is a method of the browser.
Since the request for permission can take a while, the result of the call is a promise:
When the user reacted to the prompt for permission to send messages, the promise resolves to a string that contains the decision of the user: permission is granted or denied.
When `permission` is granted, the HandleRegister method is again called: in that case, the Firebase messaging API's `getToken` method will succeed and the token will be sent to the `HaveToken` call. This call shows the token in the HTML, and sends the token to our application server:

❺ OBTAINING A TOKEN USING A WEBSITE

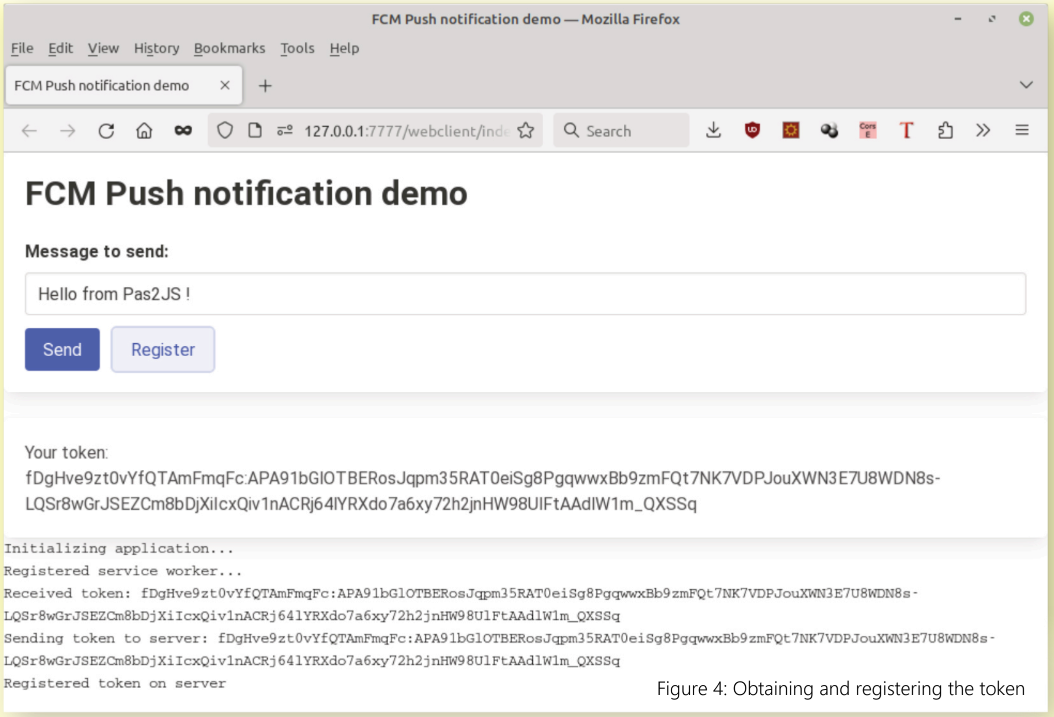
```pascal
procedure TDemoApp.HaveToken(aToken : string);
begin
  Showtoken(aToken);
  Sendtoken(aToken);
  btnSend.disabled:=False;
  btnRegister.disabled:=False;
end;
```

The `ShowToken` is easy, it sets the inner text of the DIV element with id lblToken:

```pascal
procedure TDemoApp.ShowToken(aToken : string);
begin
  pnlToken.classlist.remove('is-hidden');
  lblToken.innerText:=aToken;
  Writeln('Received token: ',aToken);
end;
```

The `SendToken` uses a JSON-RPC call RegisterSubscription to send the token to our application server :

```pascal
procedure TDemoApp.SendToken(aToken : string);

  procedure DoOK(aResult: JSValue);
  begin
    Writeln('Registered token on server');
  end;

  procedure DoFail(Sender: TObject; const aError: TRPCError);
  begin
    Writeln('Failed to register token on server: '+aError.Message);
  end;

  begin
    Writeln('Sending token to server: ',aToken);
    RPCModule.Service.RegisterSubscription(aToken,@DoOK,@DoFail);
  end;
```


FCM Push notification demo — Mozilla Firefox
File Edit View History Bookmarks Tools Help
FCM Push notification demo
127.0.0.1:7777/webclient/inde
Search
Send
Register

FCM Push notification demo

Message to send:

Hello from Pas2JS !

Your token:
fDgHve9zt0vYfQTAmFmqFc:APA91bGlOTBERosJqpm35RAT0eiSg8PgqwwxBb9zmFQt7NK7VDPJouXWN3E7U8WDN8s-LQSr8wGrJSEZCm8bDjXilcxQiv1nACRj64lYRXdo7a6xy72h2jnHW98UlFtAAdlW1m_QXSSq

```
Initializing application...
Registered service worker...
Received token: fDgHve9zt0vYfQTAmFmqFc:APA91bGlOTBERosJqpm35RAT0eiSg8PgqwwxBb9zmFQt7NK7VDPJouXWN3E7U8WDN8s-
LQSr8wGrJSEZCm8bDjXiIcxQiv1nACRj64lYRXdo7a6xy72h2jnHW98UlFtAAdlW1m_QXSSq
Sending token to server: fDgHve9zt0vYfQTAmFmqFc:APA91bGlOTBERosJqpm35RAT0eiSg8PgqwwxBb9zmFQt7NK7VDPJouXWN3E7U8WDN8s-
LQSr8wGrJSEZCm8bDjXiIcxQiv1nACRj64lYRXdo7a6xy72h2jnHW98UlFtAAdlW1m_QXSSq
Registered token on server
```

Figure 4: Obtaining and registering the token

The result of this code can be seen in *figure 4 on page 14 of this article*.
Once the token has been obtained and was successfully sent to the server, the user can use the **Send button** to send a message to himself. In a real application, the messages will of course be sent by the server in response to some external event. The event handler of the **Send button** is `HandleSend`: This method constructs a small **JSON** object with the data for the message.

❺ OBTAINING A TOKEN USING A WEBSITE

NOTE that the format of the message object is the same as the one we used to send a message using the command-line utility presented earlier. When the message object is constructed, the handler uses the `SendNotification JSON-RPC` call to send the message to our application server:

```
procedure TDemoApp.handlesend(event: TJSEvent);

  procedure DoOK(aResult: JSValue);
  begin
    Writeln('Message transferred to server for sending');
  end;

  procedure DoFail(Sender: TObject; const aError: TRPCError);
  begin
    Writeln('Failed to transfer message to server for sending: '+aError.Message);
  end;

var Msg : TJSObject;

begin
  Msg:=New([
    'title','Free Pascal FCM demo',
    'body',edtMessage.Value,
    'image','https://www.freepascal.org/favicon.png'
  ]);
  Writeln('Sending message : ',TJSJSON.stringify(Msg));
  RPCModule.Service.SendNotification(Msg,@DoOK,@DoFail);
end;
```

This concludes the interactive part of the application.
The application server exposes a small **JSON-RPC service**. As shown in previous articles about **PAS2JS** and its support for **JSON-RPC**, a unit with the proxy object for this **JSON-RPC server** can be generated automatically (*the unit is called* `service.messagingserver`).
The generated proxy object has the following declaration:

```
TMessagingService = Class(TRPCCustomService)
Protected
  Function RPCClassName : string; override;
Public
  Function SendNotification (Message : TJSObject;
                             aOnSuccess : TJSValueResultHandler = Nil;
                             aOnFailure : TRPCFailureCallBack = Nil) : NativeInt;
  Function RegisterSubscription (Token : String;
                             aOnSuccess : TJSValueResultHandler = Nil;
                             aOnFailure : TRPCFailureCallBack = Nil) : NativeInt;
end;
```

An instance of this proxy object is created on a **RPCModule** datamodule, which is implemented in the `module.messagingservice` unit:

```
TRPCModule = class(TDataModule)
 Client: TPas2jsRPCClient;
 procedure DataModuleCreate(Sender: TObject);
private
 FService: TMessagingService;
public
 Property Service : TMessagingService Read FService;
end;
```

The data module has a `TPas2JSRPCClient` component on it, and the proxy object is created in the `OnCreate` event of the **RPCModule** datamodule, and connected to the **RPCClient** object:

```
procedure TRPCModule.DataModuleCreate(Sender: TObject);
begin
  FService:=TMessagingService.Create(Self);
  FService.RPCClient:=Client;
end;
```

With that, our web application is finished.

❻ SENDING A MESSAGE FROM AN APPLICATION SERVER

In the above, we created a web page that is set up to receive and display push notifications.
The actual notifications are sent by a **HTTP application server**:
The demonstration application server is a simple HTTP application. It exposes a **JSON-RPC service**, which is implemented in a unit module.rpc. The actual handling of the **RPC** calls is implemented in the module.messaging unit, by 2 `TJSONRPCHandler` objects called `RegisterSubscription` and `SendNotification`.
The `OnExecute` event handler of the `RegisterSubscription` object is quite simple:
it extracts the token from the **JSON** data passed from the browser:

```
procedure TdmMessaging.RegisterSubscriptionExecute(Sender: TObject;
    const Params: TJSONData;
    out Res: TJSONData);
var
  Parms: TJSONArray absolute params;
  aToken : UTF8String;

begin
  If Parms.Count<>1 then
    Raise Exception.Create('Invalid param count');
  If Parms[0].JSONType<>JTString then
    Raise Exception.Create('Invalid param type for token');
  aToken:=Parms[0].AsString
  SaveToken(aToken);
  Res:=TJSONBoolean.Create(True);
end;
```

When the call to the `Savetoken` method returns, a result is sent back to the browser.
The `SaveToken` method uses a simple stringlist, which it loads from file, adds the token to and saves again in the file:

```
procedure TdmMessaging.SaveToken(const aToken : UTF8String);
var
  L : TStrings;
  FN : String;
begin
  FN:=DeviceTokensFileName;
  L:=TStringList.Create;
  try
    if FileExists(FN) then
      L.LoadFromFile(FN);
    L.Add(aToken);
    L.SaveToFile(FN);
  finally
    L.Free;
  end;
end;
```

The `DeviceTokensFileName` function returns the name of the file in which tokens must be saved, the details of this function are in the source code of this application.
The handling of the `SendNotification` message does something similar: it extracts the data from the **JSON** object passed from the client to fill the `TNotificationMessage` object:

❻ SENDING A MESSAGE FROM AN APPLICATION SERVER

```
procedure TdmMessaging.SendNotificationExecute(Sender: TObject;
               const Params: TJSONData;
               out Res: TJSONData);
var
 Parms: TJSONArray absolute params;
 Obj   : TJSONObject;
 Msg   : TNotificationMessage;

begin
 If Parms.Count<>1 then
   Raise Exception.Create('Invalid param count');
 If Parms[0].JSONType<>jtObject then
   Raise Exception.Create('Invalid notification');
 Obj:=Parms.Objects[0];
 Msg:=TNotificationMessage.Create;
 try
   Msg.Title :=Obj.Get('title',Msg.Title);
   Msg.Body  :=Obj.Get('body',Msg.Body);
   Msg.Image :=Obj.Get('image',Msg.Image);
   SendMessage(Msg);
   Res:=TJSONBoolean.Create(True);
 finally
   Msg.Free;
 end;
end;
```

When the message object is filled with the data from the webpage, it is passed on to the `SendMessage` method, and a result is sent back to the browser.
The `SendMessage` method does the actual work of sending the push notification message with the `TFCMClient` class. It starts by loading the last token from the token file created by the `RegisterSubscription` call:

```
procedure TdmMessaging.SendMessage(Msg : TNotificationmessage);
var
  Sender : TFCMClient;
  aConfig, aToken : String;

begin
  aToken:=LoadLastToken;
  Sender:=TFCMClient.Create(Self);
  try
    aConfig:=GetServiceAccountFileName;
    Sender.LogFile:=GetLogFileName;
    Sender.InitServiceAccount(aConfig,'');
    Sender.OnNewBearerToken:=@HandleNewAccessToken;
    if FileExists(AccessTokenFile) then
      Sender.BearerToken.LoadFromFile(AccessTokenFile);
    Sender.Send(Msg,aToken);
  finally
    Sender.Free;
  end;
end;
```

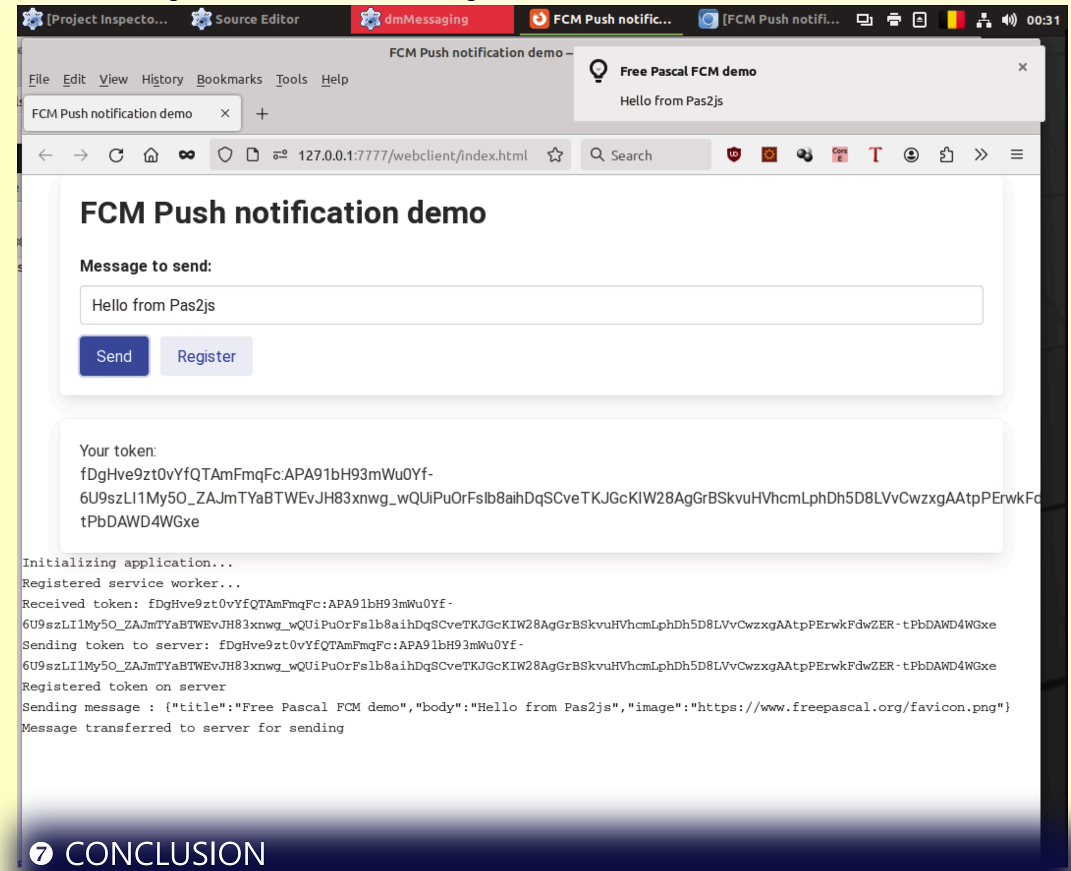
The `HandleNewAccessToken` method (*used to save the access token*) is identical to the one in our **command-line utility**.
All that remains to be done is to add the units that set the default **WebClient** class to use.
After that, our application is finished.
It should be clear that other than saving and loading the token from a file, this application server code is not substantially different from the code in the command-line utility we constructed earlier.

❻ SENDING A MESSAGE FROM AN APPLICATION SERVER

To run this example, you must follow the following steps:

- Make sure the service account file is located next to the HTTP server program (*messageserver*) with the correct name: **`messagserver-serviceaccount.json`**.
- Start the messageserver HTTP server program.
- Make sure the **`config.js`** file is properly set up as described above, as well as the firebase Javascript files.
- Start the web-based client program in the browser.
  To do so, for example in Lazarus, just press F9 to run it, and it should open in the browser.
- In the **Firefox** browser, make sure the developer tools console is opened (**press F12**), and set the '**Enable service workers over HTTP** (*when toolbox is open*)' in the toolbox settings.
  This step is only necessary for testing:
  if you host the page on a HTTPS website, then service workers will automatically be allowed.
- Press the register button in the website.
  You should see a confirmation message as in *figure 4 on page 16 of this article*.
- Type some nice message and press 'Send'.

The result should look like *figure 5 on page 17*. Note that due to the particular desktop manager of the author, the icon is not shown in the picture. If you noted the token, you should also be able to send a message to the same browser using the command-line tool.


[Project Inspecto...
Source Editor
dmMessaging
FCM Push notific...
[FCM Push notifi...
00:31
FCM Push notification demo –
File Edit View History Bookmarks Tools Help
FCM Push notification demo
127.0.0.1:7777/webclient/index.html
Search

Free Pascal FCM demo
Hello from Pas2js

FCM Push notification demo

Message to send:

Hello from Pas2js

Send Register

Your token:
fDgHve9zt0vYfQTAmFmqFc:APA91bH93mWu0Yf-
6U9szLI1My5O_ZAJmTYaBTWEvJH83xnwg_wQUiPuOrFslb8aihDqSCveTKJGcKIW28AgGrBSkvuHVhcmLphDh5D8LVvCwzxgAAtpPErwkFd
tPbDAWD4WGxe

```
Initializing application...
Registered service worker...
Received token: fDgHve9zt0vYfQTAmFmqFc:APA91bH93mWu0Yf-
6U9szLI1My5O_ZAJmTYaBTWEvJH83xnwg_wQUiPuOrFslb8aihDqSCveTKJGcKIW28AgGrBSkvuHVhcmLphDh5D8LVvCwzxgAAtpPErwkFdwZER-tPbDAWD4WGxe
Sending token to server: fDgHve9zt0vYfQTAmFmqFc:APA91bH93mWu0Yf-
6U9szLI1My5O_ZAJmTYaBTWEvJH83xnwg_wQUiPuOrFslb8aihDqSCveTKJGcKIW28AgGrBSkvuHVhcmLphDh5D8LVvCwzxgAAtpPErwkFdwZER-tPbDAWD4WGxe
Registered token on server
Sending message : {"title":"Free Pascal FCM demo","body":"Hello from Pas2js","image":"https://www.freepascal.org/favicon.png"}
Message transferred to server for sending
```

❼ CONCLUSION

In this article we showed that sending push notifications to a user is not so difficult to do. In fact, more time is spent on setting up all the necessary files and configuring the project on firebase, than on actually coding the application. The sending of messages made use of **FCM - Firebase Cloud Messaging.** It is possible to do the same without Firebase, by using the WebPush protocol built-in in the browser. We'll leave the discussion of that to a future contribution.

The opening session was kicked off by greetings from Dieter Kranzmüller (BADW-LRZ), Uwe Heitmann from DLR-PT, and Jorge Gasos and Stergios Tsiafoulis from DG CONNETC. OpenWebSearch.EU has kicked-off with a consortium meeting in Berlin. 46 participants got together in a hybrid meeting format to exchange ideas and make plans for the successful execution of the project. The meeting was hosted by DLR-PT in a new building in Berlin-Südkreuz.

Open WebSearch.eu

NGI OPENWEBSEARCH.EU BY DETLEF OVERBEEK

INTRODUCTION

14 well-known European computing and research institutes have joined forces to create an open internet search infrastructure in Europe.
The project contributes to **Europe's digital sovereignty and can protect the world's freedom values** - especially in Europe - and promote an **open, people-centric search engine market.**

In September 2022, partners including computing centres and universities launched the EU **OpenWebSearch.eu** project. This is the first project funded by the European Union to get the future web search engine off the ground.

At this point, the OpenWebSearch.eu initiative was officially launched, funded by the EU to enable web search.
Concerns about the **imbalance in the search engine market** formed the basis for the project: **Google's** web indexes, such as the Start page, or **Bing's** Start pages, used by **DuckDuckGo**, **Ecosia**, **MetaGer, Neeva, Qwant and You.com.**
In practice, the answer results are limited to these names and the world is therefore incomplete. This increases distrust in society.
Besides the two US search indexes, there are only two other comparable providers worldwide after Yahoo's acquisition by Bing (Microsoft), which are Yandex from Russia and Baidu from China.
The latter two are clearly less important for Europe, mainly because of their language differences and unclear presentation of facts.

Information as a public good, with free, unbiased and transparent access, is no longer under public control with them. This lack of an open search environment jeopardises our freedom and impoverishes the innovative power of societies, technology, research institutes and the economy.

OBJECTIVE

Over the next few years, researchers will seek to develop the core of a **European Open Web Index** (**OWI**) as the basis for a new **Internet Search Mechanism** in Europe.

It will also lay the foundations for an open and extensible European **Open Web Search and Analysis Infrastructure (OWSAI),** based on European values, principles, legislation and standards.

- It could thus be a large - **freely accessible to the public - Home Page.**
  In practice, it appears that the public uses the sites mainly as search engines.
  Which is what they originally were: search engines.
- **There will then no longer be the Search engine Manipulation Effect:**
- **The fear of (large-scale) manipulation of data and realities (fake news)**,
  turning results and reports into misinformation, or false truths.
  Opinion formation can be influenced by this, which only serves the interests of dictators and undemocratic miscreants. And that, in turn, affects democracy and our freedom.
  Very important, then.

The **intention is to develop a web index** containing **half of all texts published on the internet.**
The partners involved expect a storage requirement of about five petabytes (five million gigabytes).
Compared to Google or Bing's indexes, this is a smaller database.
The established competition each has hundreds of petabytes of texts, image files, multimedia, usage data and log files from the internet.

Under no circumstances should it become a repeat of the **Google** concept or making it a **European Google**, according to **Michael Granitzer**, professor of **Data Science University of Passau**, who is coordinating the **OpenWebSearch** project.

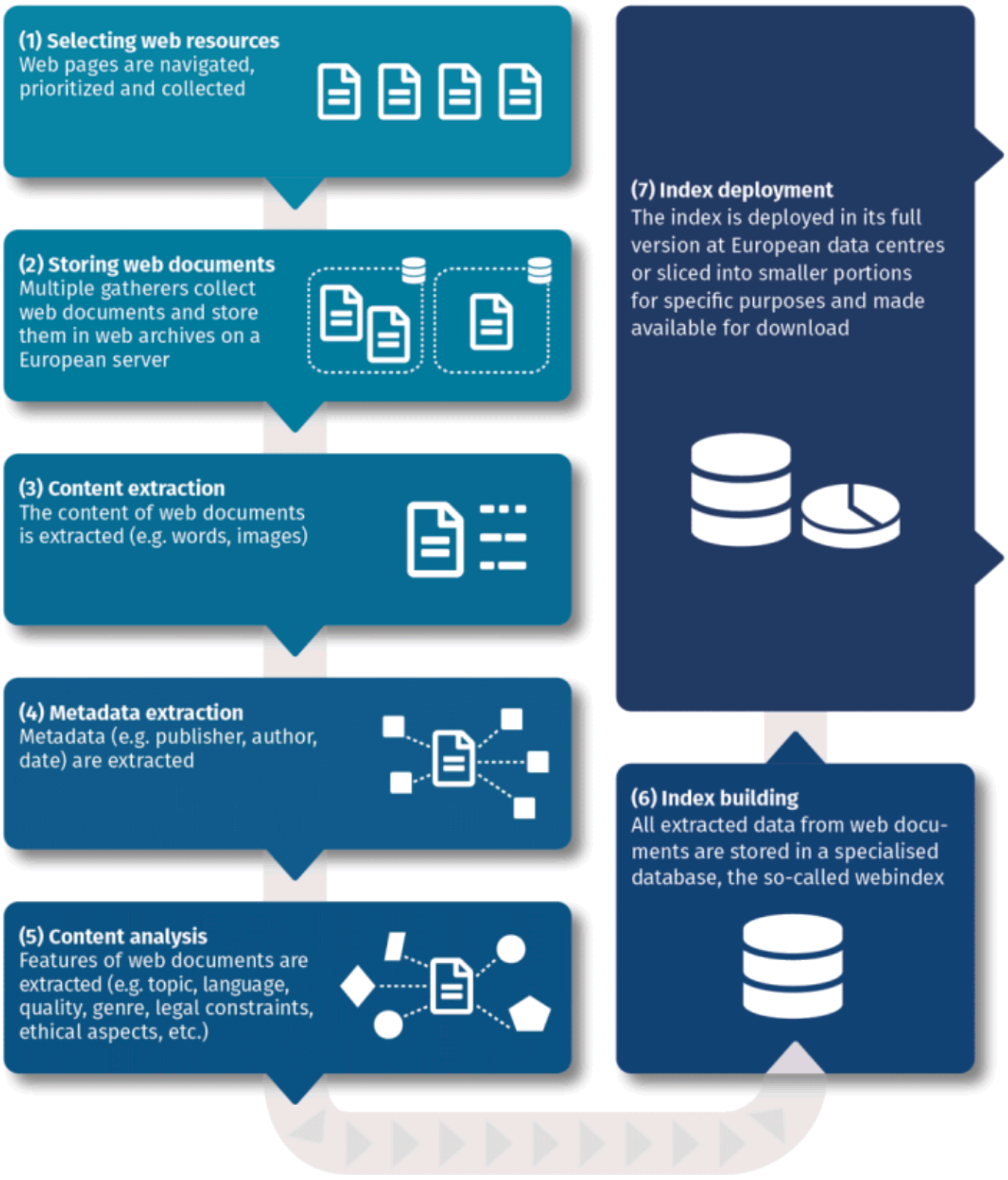
LIST OF PROJECT PARTNERS
1. University of Passau, Germany (uni-passau.de)
2. Leibniz Supercomputing Centre of Bavarian Academy of Sciences and Humanities, Germany (lrz.de)
3. Stichting Radboud Universiteit, Netherlands (ru.nl)
4. Leipzig University, Germany (uni-leipzig.de)
5. Graz University of Technology, Austria (tugraz.at)
6. Deutsches Zentrum für Luft- und Raumfahrt, Germany (dlr.de)
7. VSB – Technical University of Ostrava, IT4Innovations, Czech Republic (www.vsb.cz)
8. European Organization for Nuclear Research – CERN, Switzerland (home.cern)
9. Open Search Foundation, Germany (opensearchfoundation.org)
10. A1 Slovenija, telekomunikacijske storitve, d. d., Slovenia (a1.si)
11. CSC-Tieteen Tietotekniikan Keskus Oy, Finland (csc.fi)
12. Stichting Nlnet, Netherlands (nlnet.nl)
13. Bauhaus-Universität Weimar, Germany (uni-weimar.de)
14. SUMA-EV – Association for Free Access to Knowledge, Germany (suma-ev.de)

The project involves setting up an infrastructure with which search engines and other services can work.

A set-up like Google's is certainly not the intention.
Rather, it will have to grow like, for example, **Wikipedia**, which contained a small core and then quickly grew in size.
Currently, a total of 14 project partners are developing crawling techniques.

Index Generation

Web resources are selected and retrieved, their content and metadata are analysed, and all data stored in the index database.



(1) Selecting web resources
Web pages are navigated, prioritized and collected

(2) Storing web documents
Multiple gatherers collect web documents and store them in web archives on a European server

(3) Content extraction
The content of web documents is extracted (e.g. words, images)

(4) Metadata extraction
Metadata (e.g. publisher, author, date) are extracted

(5) Content analysis
Features of web documents are extracted (e.g. topic, language, quality, genre, legal constraints, ethical aspects, etc.)

(6) Index building
All extracted data from web documents are stored in a specialised database, the so-called webindex

(7) Index deployment
The index is deployed in its full version at European data centres or sliced into smaller portions for specific purposes and made available for download

Open WebSearch .eu

Search Applications

A user search request will be answered by a search application that makes use of the open web index.

(a) Selecting web documents
Web documents are selected that fit to the user search request

(b) Ranking web documents
The selected Web documents are sorted (ranked) according to their assumed relevance for the user

(c) Purpose-specific search
An application with user interface enables the search for general or specific purposes

(d) User searches and receives result
The user is supported to better understand the search process

Data Products

Knowledge representation models will be created using the open web index, in order to be used by any agent and for many applications

Building knowledge graphs
Using the extracted information from web documents, a knowledge graph is created that supports specific search requests

Building AI Language Models
Creation of different types of language models by using Web documents

Any agent, multiple applications
Language models and knowledge graphs can be used by any agent (or application)

• • •

Technically:
crawling focuses on collecting content from a web page, while indexing has a different function.
Indexing namely focuses on storing and organising the content.
During this process, its relevance to certain searches is determined.

These partners - where you can find the addresses logos in this article - select **metadata*** that the index should contain and thus design a decentralised distribution of the **OWI** across different servers and locations in Europe.
(**Metadata** *are defined as the data that provide information about one or more aspects of the data; they are used to summarise basic information about data that can make tracking and working with specific data easier. Some examples are: Mode of creation of the data. Purpose of the data. Time and date of creation.*)

PARTICIPANTS (PARTNERS):
Infrastructure partners participating include
the **Leibniz Supercomputing Centre** in Munich,
the **CSC** in Espoo, Finland, which operates Europe's largest supercomputer,
the **Czech National Supercomputing Centre IT4Innovations** and
CERN near Geneva. Other partners include the
Dutch Radboud University Foundation and
Stichting Nlnet.

Search
Suche
Zoeken
Ricerca
Vyhledávání
Iskanje
Recherche
Etsi
Composition: openwebsearch.eu / using images by NASA (europe_dnb_2012_lrg.jpg), Unsplash (christopher-burns-dzejyfCAzlA-unsplash)

DATA COLLECTION

By targeting the data orchestrated through this index page, you could yourself innovate in this area by, for example, diversifying the use of the mechanism - as all problems should be solved - make them smaller:

by **using various categories of search engines** that are much better and faster at providing incisive answers and even **unsolicited predictions** in their own areas.
If it turns out that there are a large number of **bird flu infections** on the web, if they then also turn out to be **transmissible** (*preferably collected through scientific knowledge*) then as a society you can respond very quickly.
That AI can be involved in this, seems obvious, but it must be done under strict control.
And Europe is ready for that.

Moreover, it is possible to **develop new language facets using an extensive index** based mainly on European sources. In particular, smaller languages or language combinations such as **Lithuanian**, **Basque** and **Frisian**, which are quickly ignored by globally-oriented services, can benefit.

Many new possibilities loom large:
take stock of the actual inflow of people outside Europe and give it a stage through this, allowing for better control of the spread and, above all, checking which individuals we need, so that we have a much clearer truth finding, all **because this is an index related to Europe**.
I see an infinite number of benefits.

Composition: openwebsearch.eu / using images by NASA (europe_dnb_2012_lrg.jpg), Unsplash (christopher-burns-dzejyfCAzIA-unsplash), OpenClipart-Vectors from Pixabay (1293096)
Open Web Search

POLITE CRAWLERS:

The **web crawlers** used are expected to be '**neat and above all respectful**'.
This has a good reason.
According to **Stefan Voigt**, the chairman of the **Open Search Foundation** that introduced the **OpenWebSearch** project, **web crawling accounts for about 30 to 40 per cent of the total network load**, if streaming* is **not** included.

(***Streaming media** *is multimedia that can be played with an offline or online media player. Technically, the stream is delivered and consumed in a continuous manner by a client, with little or no intermediate storage in network elements. Streaming refers to the delivery method of content rather than the content itself.* **It consumes lots of energy**).

This is a significant cost for web hosts.
Moreover, a website is easily overloaded if all HTML pages are retrieved in parallel (simultaneously) with multiple servers.
Due to an implementation error, this happened when researchers at the project crawled Bank of America's pages. The bank misunderstood that as a denial-of-service attack and blocked the servers involved.

With so-called **Crawling Politeness Rules**, such misunderstandings should in principle be avoided. For example, never more than **one request per second** should be started and the page restrictions of the operator's robots.txt file should of course be observed.

This allows website operators to ask '**bots**' **not to visit certain parts of their website.**
Analyzing the crawls already reveals **another drawback compared to the Google-dominated index:** the robust editing and partitioning of HTML files is a problem that cannot simply be solved technically, but **only in collaboration with website operators.**

Google offers them the **Google Search Console** for that, an analytics tool that allows them to perform **search engine optimization** (**SEO**) on their own and thus adapt their pages specifically to the **Google bot**.

There, webmasters can, for example, see at a glance whether the main text of a webpage is where Google expects it to be.

Does the analysis produce a complete parse-tree?
What keywords does the **Google bot recognise** on web pages?

Most website owners have pages that contain opinions and can lead to language AI's with bias:
(*Bias is a disproportionate weight in favour or against an idea or thing, usually in a way that is inaccurate, closed, biased or inappropriate*).

In time, a public, **transparently built web index** could **completely change the SEO landscape**.
These days, almost every entrepreneur is trying to optimize their website for a high ranking on search results with Google.

If a large number of specialised search services replaced that monopoly,
the situation would change fundamentally.
The best thing would be if website owners and web users concentrate on creating their site's content.

**When web search is no longer dominated by a monopolist,
all that matters is preparing content in the most structured way possible
and supporting it with meaningful metadata.**

Most website owners have pages that contain opinions and can lead to language AI's with bias: (*Bias is a disproportionate weight in favour or against an idea or thing, usually in a way that is inaccurate, closed, biased or inappropriate*).
In time, a public, **transparently built web index** could **completely change the SEO landscape**.
These days, almost every entrepreneur is trying to optimize their website for a high ranking on search results with Google.
If a large number of specialised search services replaced that monopoly, the situation would change fundamentally. The best thing would be if website owners and web users concentrate on creating their site's content.
When web search is no longer dominated by a monopolist, all that matters is preparing content in the most structured way possible and supporting it with meaningful metadata.

THERE IS NO USER ANALYSIS

The selection of **search factors** to be included in the new web index is still open and under intense discussion. Besides **content usage rights**, the partners mainly discuss content factors such as content quality, genres and a page rank weighting of link structures (URLs).

There are also technical factors such as **response times.**
Exciting research projects concern the idea of georeferencing, (**indicating location via coordinates, for example**) that enables **web services for individual cities or communities.**
However, the **spatial classification** of web links requires proper analysis first, as America lies not just in the Netherlands.

A VERY IMPORTANT DIFFERENCE FROM CURRENT LARGE INDEXES:
user-clicks and other user analytics do not reach the OWI.
This is hardly possible, because the project partners do not intend to operate search engines themselves and thus collect user data.
'**Web services that build on the OWI can of course always record the search term and user behaviour on their platform**,' explains the technical adviser to the **non-profit organisation Suma**, which operates the **MetaGer** metasearch engine.
*(*MetaGer is a metasearch engine focused on protecting users' privacy. Based in Germany, and hosted as a cooperation between the German NGO 'SUMA-EV - Association for Free Access to Knowledge' and the University of Hannover, the system is built on 24 small-scale web crawlers under MetaGer's own control. In September 2013, MetaGer launched MetaGer.net, an English-language version of their search engine*).
To some extent, **this is even necessary, for instance to be able to block mass searches** from spammers or to analyse and improve its own service.
But always, users will be free to decide to whom they entrust their usage data.

SECURELY DISTRIBUTED

Because of the **critical infrastructure** for Europe, it is planned to distribute the **OWI** across several countries. That way, like national libraries, the **OWI** could be housed in different data centres. Indexes could be developed in different countries with a regional focus, e.g. by language of origin.

Moreover, different organizations could take on the **maintenance of different sub-indexes** and host them, e.g. an index for **geosciences** or an **index for financial** markets, for **green distribution and planting and water flow**. **One of the problems in most countries is a poor or absent overview of all pipeline data lying three-dimensionally in the soil** (*would be a huge advance for urban planning, as financial and social damage can be anticipated and prevented*).

CERN, for example, has already expressed interest in compiling and managing all information on particle physics. The project will not remain solely in scientific hands.
There are project funds and people need economic players.
It is not only necessary to have a researcher's mind; you also need to be entrepreneurial and have business ideas to create, for example, new search engines and services based on the OWI.
Because the medium-term growth of the web index is only possible if the licensing revenue pays for the infrastructure costs.

POOL FOR LANGUAGE MODELS (LLCs and real languages)

Right at the start of the project and thus even before the hype around ChatGPT,
the partners considered the **Open Web Index**, with its focus on European content and languages, to be a data pool for specialised language models.
New search engines could also immediately use those models as an interface for searches.

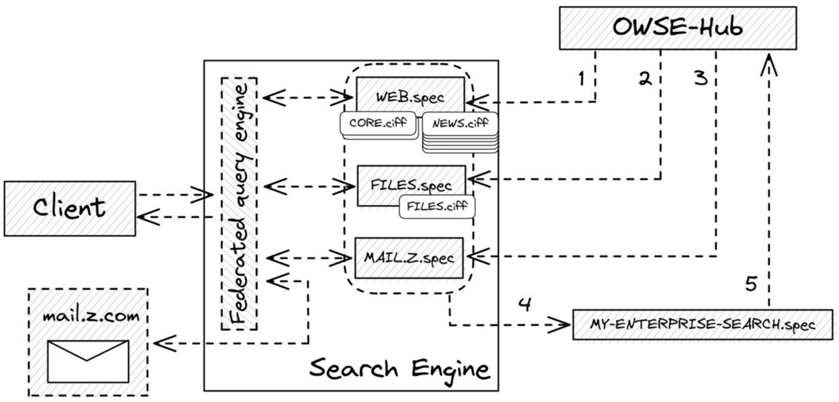
Users are usually not looking for links, but for answers to their questions or even suggestions for solutions.

That speaks in favour of using chatbots.

The search engine **Bing**, with its **ChatGPT interface**, and **Google, with its Gemini**, seem poised to overtake that approach on the right.
Project coordinator **Granitzer** also sees the search engine **You.com** as a successful example of how an **AI chat bot** can capture search queries and describe result links.

However, one cannot rely on AI's answers at the moment.
Moreover, studies on the cost and scalability of such systems are yet to be done.

Language models as an interface will have to be integrated into search engines in the future. because this technology provides a way to process search queries in natural language or like a chat bot and summarize the results.
Therefore, they are the better interface for starting an overview.



```
1: owse pull web-search
2: owse pull companyX/files
3: owse pull companyZ/mail
4: owse build my-enterprise-search
5: owse push my-enterprise-search
```

NEW SERVICES ON THE INTERNET

The OpenWebSearch project is looking for new partners to add their expertise.
There are already discussions with specialised services such as a vocabulary lexicon, which recognizes new word creations, metaphors and idioms automatically and live using an up-to-date, freely accessible web index.

Another project, **Europe Media Monitor,** examines trending topics, current events and developments on the internet.
`https://knowledge4policy.ec.europa.eu/online-resource/europe-media-monitor-emm_en`
New services could, for example, list the pros and cons of controversial topics instead of showing search results as Google and its competitors do.
This is already the case on the **Args.me** website.(very hard to reach!)
Moreover, the **OWI** could form the basis for search services specific to a particular topic.

Even the creation of **dedicated search engines for mobile phones** could be the result of the search index limitation.

As a result,
users could search locally without access to the internet
and they would only have to reveal their user data on their mobile device.

You can imagine yourself as a search engine hub near the **OWI** one day.
The user chooses his **areas of interest** and can use a number of features, such as full-text search.
He limits the results to sources that are particularly credible or popular.
In the end, **he receives a search engine configured according to his preferences** after he submits his form.

CRUCIAL INFRASTRUCTURES

The EU's OpenWebSearch project will initially last until September 2025.
In between, the partners aim to build the five petabyte base index.
After that, the focus will be on sustainable infrastructure funding, probably through additional EU funding.

CONCLUSION

The Open Web Index is an essential infrastructure for Europe's digital sovereignty.
It is expected by the project partners to create transparent structures on the internet.
The envisaged **European Web Index** is intended to create more diversity and is likely to be particularly useful for those who want to offer only the best and most reliable information on their websites.

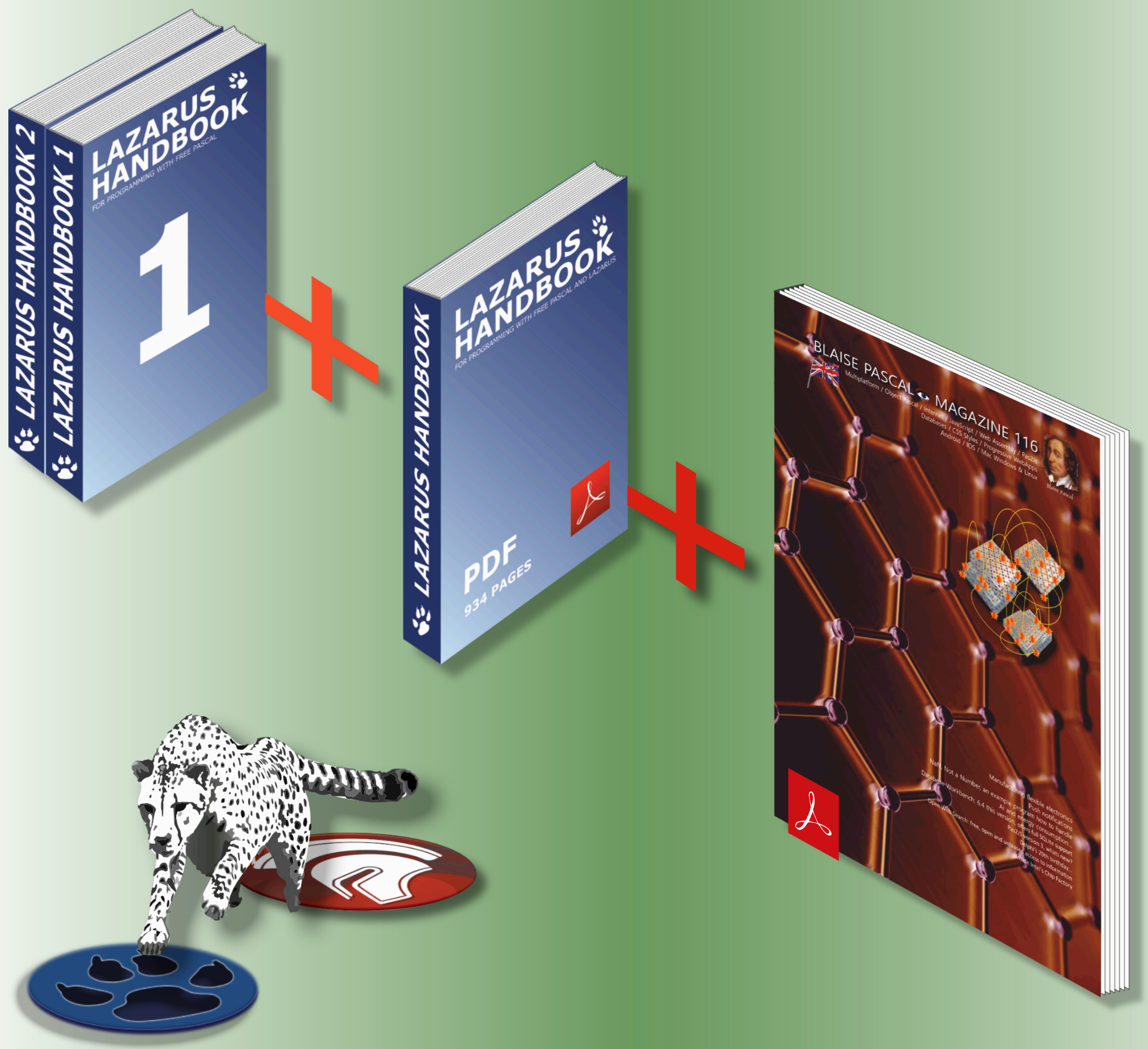
The content of this article has been composed by articles originating from these addresses:
https://asistdl.onlinelibrary.wiley.com/doi/10.1002/asi.24818 /
https://openwebsearch.eu / https://openwebsearch.eu/community/3rdparty-calls/call3/
https://open-console.eu/ https://zenodo.org/communities/owseu/records?q=&l=list&p=1&s=10&sort=newest /
https://openwebsearch.eu/partners/ as well as other articles not mentioned here.

ADVERTISEMENT

BLAISE PASCAL MAGAZINE 114/115

Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js / Databases / CSS Styles / Progressive Web Apps Android / IOS / Mac / Windows & Linux

Blaise Pascal



LAZARUS HANDBOOK POCKET+PDF+ DOWNLOAD MAGAZINE SUBSCRIPTION

EX VAT AND SHIPPING PRICE: € 75,00

https://www.blaisepascalmagazine.eu/product-category/books/

Donate for Ukraine and get a free license at:
https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

If you are from Ukrainian origin you can get a free Subscription for Blaise Pascal Magazine, we will also give you a free pdf version of the Lazarus Handbook. You need to send us your Ukrainian Name and Ukrainian email address (*that still works for you*), so that it proofs you are real Ukrainian. please send it to editor@blaisepascal.eu and you will receive your book and subscription

BLAISE PASCAL MAGAZINE

Multi platform /Object Pascal / Internet / JavaScript / Web Assembly / Pas2Js / Databases / CSS Styles / Progressive Web Apps Android / IOS / Mac / Windows & Linux

Blaise Pascal

COMPONENTS DEVELOPERS 4

Donate for Ukraine and get a free license at:
https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

COMPONENTS DEVELOPERS 4

Donate for Ukraine and get a free license at:
https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

kbmMW Professional and Enterprise
NEW EDITION V. 5.23
kbmMemTable
NEW EDITION V. 7.99.00
Standard and Professional Edition

5.23.00 is a release with containing new stuff, refinements and bugfixes, **openSSL v3 support,** WebSocket support, further improvements to SmartBind, new high performance hashing algorithms, improved RemoteDesktop sample and much more.
This release requires the use of **kbmMemTable** v. 7.98.00 or newer.

- RAD Alexandria supported
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OS X client and server support
- Native high performance 100% developer defined application server
- Full support for centralised and distributed load balancing and fail-over
- Advanced ORM/OPF support including support of existing databases
- Advanced logging support
- Advanced configuration framework
- Advanced scheduling support for easy access to multi thread programming
- Advanced smart service and clients for very easy publication of functionality
- High quality random functions.
- High quality pronounceable password generators.
- High performance LZ4 and J peg compression
- Complete object notation framework including full support for YAML, BSON, Messagepack, J SON and XML
- Advanced object and value marshalling to and from YAML, BSON, Messagepack, JSON and XML
- High performance native TCP transport support
- High performance HTTPSys transport for Windows.
- CORS support in REST/HTML services.
- Native PHP, Java, OCX, ANSI C, C#, Apache Flex client support!

kbmMemTable is the fastest and most feature rich in memory table for Embarcadero products.

- Easily supports large datasets with millions of records
- Easy data streaming support
- Optional to use native SQL engine
- Supports nested transactions and undo
- Native and fast build in M/D, aggregation/grouping range selection features
- Advanced indexing features for extreme performance

- New: full Web-socket support.
  The next release of kbmMW Enterprise Edition will include several new things and improvements.
  One of them is full Web-socket support.
- New I18N context sensitive internationalisation framework to make your applications multilingual.
- New ORM LINQ support for Delete and Update.
  Comments support in YAML.
- New StreamSec TLS v4 support (by StreamSec)
  Many other feature improvements and fixes.

Please visit http://www.components4developers.com
for more information about kbmMW

- High speed, unified database access (35+ supported database APIs) with connection pooling, metadata and data caching on all tiers
- Multi head access to the application server, via REST/AJAX, native binary, Publish/Subscribe, SOAP, XML, RTMP from web browsers, embedded devices, linked application servers, PCs, mobile devices, Java systems and many more clients
- Complete support for hosting FastCGI based applications (PHP/Ruby/Perl/Python typically)
- Native complete AMQP 0.91 support (Advanced Message Queuing Protocol)
- Complete end 2 end secure brandable Remote Desktop with near realtime HD video, 8 monitor support, texture detection, compression and clipboard sharing.
- Bundling kbmMemTable Professional which is the fastest and most feature rich in memory table for Embarcadero products.

COMPONENTS 4 DEVELOPERS

ADVERTISEMENT