BLAISE PASCAL MAGAZINE 91

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

Blaise Pascal

BLAISE PASCAL MAGAZINE 91

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

Blaise Pascal

CONTENT

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 - Netherlands

Contributors

Stephen Ball
`http://delphiaball.co.uk`
`@DelphiABall`
		Peter Bijlsma -Editor
		`peter @ blaisepascal.eu`

Dmitry Boyarintsev
`dmitry.living @ gmail.com`
	Michaël Van Canneyt,
	`michael @ freepascal.org`
		Marco Cantù
		`www.marcocantu.com`
		`marco.cantu @ gmail.com`

David Dirkse
`www.davdata.nl`
`E-mail: David @ davdata.nl`
	Benno Evers
	`b.evers`
	`@ everscustomtechnology.nl`
		Bruno Fierens
		`www.tmssoftware.com`
		`bruno.fierens @ tmssoftware.com`

Holger Flick `holger @ flixments.com`		

Primož Gabrijelčič
`www.primoz @ gabrijelcic.org`
	Mattias Gärtner
	`nc-gaertnma@netcologne.de`
		Peter Johnson
		`http://delphidabbler.com`
		`delphidabbler @ gmail.com`

Max Kleiner
`www.softwareschule.ch`
`max @ kleiner.com`
	John Kuiper
	`john_kuiper @ kpnmail.nl`
		Wagner R. Landgraf
		`wagner @ tmssoftware.com`

Vsevolod Leonov
`vsevolod.leonov@mail.ru`
		Andrea Magni `www.andreamagni.eu`
		`andrea.magni @ gmail.com`
		`www.andreamagni.eu/wp`

	Paul Nauta PLM Solution Architect CyberNautics `paul.nauta @ cybernautics.nl`	**Kim Madsen** `www.component4developers.com`
Boian Mitov `mitov @ mitov.com`		**Jeremy North** `jeremy.north @ gmail.com`

Detlef Overbeek - Editor in Chief
`www.blaisepascal.eu`
`editor @ blaisepascal.eu`
	Howard Page Clark
	`hdpc @ talktalk.net`
		Heiko Rompel
		`info @ rompelsoft.de`

Wim Van Ingen Schenau -Editor `wisone @ xs4all.nl`	**Peter van der Sman** `sman @ prisman.nl`	**Rik Smit** `rik @ blaisepascal.eu`

Bob Swart
`www.eBob42.com`
`Bob @ eBob42.com`
	B.J. Rao
	`contact @ intricad.com`
		Daniele Teti
		`www.danieleteti.it`
		`d.teti @ bittime.it`

Anton Vogelaar `ajv @ vogelaar-electronics.com`	**Robert Welland** `support @ objectpascal.org`	**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`

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

Correctors
Howard Page-Clark, Peter Bijlsma

Trademarks All trademarks used are acknowledged as the property of their respective owners.
Caveat Whilst we endeavour to ensure that what is published in the magazine is correct, we cannot accept responsibility for any errors or omissions. If you notice something which may be incorrect, please contact the Editor and we will publish a correction where relevant.
Subscriptions (2019 prices)

	Internat. excl. VAT	**Internat. incl. 9% VAT**	**Shipment**
Printed Issue ±60 pages	€ 155,96	€ 250	€ 80,00
Electronic Download Issue 60 pages	€ 64,20	€ 70	———
Printed Issue inside Holland (Netherlands) ±60 pages	———	€ 240,00	€ 70,00

Member and donator of WIKIPEDIA
Member of the **Royal Dutch Library**

Subscriptions can be taken out online at `www.blaisepascal.eu` or by written order, or by sending an email to `office@blaisepascal.eu`
Subscriptions can start at any date. All issues published in the calendar year of the subscription will be sent as well.
Subscriptions run 365 days. Subscriptions will not be prolonged without notice. Receipt of payment will be sent by email.
Subscriptions can be paid by sending the payment to:
ABN AMRO Bank Account no. 44 19 60 863 or by credit card or Paypal
Name: Pro Pascal Foundation-Foundation for Supporting the Pascal Programming Language (Stichting Ondersteuning Programeertaal Pascal)
IBAN: NL82 ABNA 0441960863 BIC ABNANL2A VAT no.: 81 42 54 147 (Stichting Programmeertaal Pascal)
Subscription department
Edelstenenbaan 21 / 3402 XA IJsselstein, The Netherlands
Mobile: + 31 (0) 6 21.23.62.68 `office@blaisepascal.eu`

From your editor

Happy New Year!
And it certainly looks like it…
We probably will be freed of Corona and there are coming some very good developments from Science, for developers there is the new rolable screen which we will soon see released and Smart Phones with enlarging size.

I predicted the Pencil–Smart Phone some years ago and here are now the first steps (page 16).
We will probably be forced to develop more and more for the web, desktop will become a minor issue. Because of this we now have great news about Pas2JS, WebCore and Lazarus for Atom or Visual Studio Code.

In this issue we have several articles about this all and offer a number of apps that you can build or run: on the web as well on the desktop. The basic components you need are already available.
For Lazarus I can tell you we are developing a special form (Martin Friebe from the Lazarus team does that) which is capable of WYSIWYG (what you see is what you get) so you can create on your desktop applications that have already the final looks and feel of the web.
He is building an Object Inspector for this form which is one with a big difference: your normal Object Inspector aims at one component but this one is made for the web and you need to be able to apply it to all of the objects on the form.
I hope we will be able to show this during spring for the first time.

Speaking about that, I want to start organizing a first meeting again. As soon we see a possibility for this we'll alert you!
At the end of this coming year we hop to publish our hundredth issue and we'll throw a party about that.

Now there still is very good news for our future:
a Dutch company was capable of creating Hydrogen as a powder (page8). That solves a lot of problems.
Just think of it… They even plan for batteries for notebooks that are exchangeable or to be refuelled. No acidic batteries any more -no more pollution, no more waist.
It will take some time but well get there…

Lets start to build the apps of the future!

To begin with this issue…

Readers Write

Robert Evans
Happy New Year!
Thank you for yet another very interesting edition of Blaise Pascal Magazine.

I read your article on **FastReport VCL 6.9** and noted your comment about needing to start Delphi via "Run as Administrator". I have been using FastReport for over ten years and can confirm that the issue that you encountered is actually much older than just version 6.9.

The underlying problem is that the default install folder suggested by the FastReport installer is under "`C:\Program Files (x86)`". In modern versions of Windows (i.e. Vista and successors) such a location is protected by Windows UAC and so cannot be written to by any program that does not have elevated privilege. Unless the project Output Directory path is manually overridden by the user, this restriction prevents the FastReport demos from being recompiled. From time to time it can also raise a problem while creating one's own programs - i.e. whenever a project Build causes a recompile of FastReport's own .bpl files.

However the solution is quite simple: just manually uninstall FastReport via the Start Menu and then re-run the installer but this time choose to override the suggested installation folder to select a more suitable location. I use a folder below "`C:\Users\Public\Public Documents\FastReport`", but the actual location is not critical as long as it is writeable when needed.

Kind regards,

Robert Evans, Director
Lichfield Technology Limited

Answer: Denis Zubov, from FastReport told me that they are going to change this for future

Wyatt Wong
You may consider to enhance the login with two-factor authentication by making use of Google Authenticator app

Best Regards,
Wyatt Wong

Answer: that is surely something we might implement in the coming year

Diego José Muñoz Carbajo
Hello, I don't know where contact respect your article 'compiled date'.
Just me: the editor: editor@blaisepascal.eu
Is not easy (and 'modern') simply to use the 'new' unit IOUtils, so the line must be:
```
TFile.GetLastWriteTime(application.exename)
```

Thx, Good job

Diego José Muñoz Carbajo
Programador. Freelance.
Ing. técnico informática gestión. EUIT. Universidad Politécnica de Madrid.

Answer: It depends. the `GetLastWriteTime` *is not the same as the compiletime. I haven't found anything like compiletime or compiledate in that unit. This is* `FileAge`.
```
Compiledate:=DateTimeToStr(FileDateToDateTime(FileAge(ExtractFileName(Application.
ExeName))));
```

From our Technical advisor: Cartoons from Jerry King

"We'll have to run more tests, but we think he's tried memorizing so many passwords his brain locked up."

Hydrogen as a powder: the energy problems are solved : H2 Fuel

https://h2-fuel.nl/portfolio-item/automotive-sector/?lang=en

Introduction of H2FUEL

H2Fuel is a patented technology for the production, storage and release of hydrogen. For its production, no electrolysis is required. The hydrogen is stored under normal atmospheric conditions in a powder. Release takes place without additional energy, using ultrapure water. Not only is one hundred per cent of the hydrogen stored in the powder released but, as a bonus, the same amount of hydrogen is released from the water, as well.

In dry powder form, the hydrogen can be stored for an unlimited period, is in energy terms the maximum attainable result, has no safety risks and, throughout the production process from production through consumption, features no harmful emissions at all. Once the hydrogen has been issued, the residual substances can be returned to the powder state with hydrogen stored in them: this makes H2Fuel the world's first circular fuel. H2Fuel can be deployed in all sectors of society and the economy and, as a result, forms by far the preferable alternative to both fossil fuels and other sustainable alternatives.

Detail:

The powder referred to above is sodium borohydride ($NaBH_4$). Each molecule of sodium borohydride contains 4 hydrogen atoms (4H). Two molecules of water (H_2O) also contain four hydrogen atoms ($2H_2$).

Ultrapure water is water from which all interfering substances have been filtered out.

Some of the required water comes from the fuel cell and is filtered.

All of the basic substances and filtration installations needed are commercially available.

One cubic metre of powder contains 9 MWh of energy.

Unpacking process

To make it pumpable, sodium borohydride, partially diluted with ultrapure water, is introduced into in a mixing chamber. Very lightly acidified ultrapure water is also introduced. Instead of acidification, a catalyst can also be utilised, depending on the requirements of use. When these ingredients meet, a natural exothermic reaction takes place, such that four hydrogen atoms split off from the sodium borohydride ($NaBH_4$), and hydrogen gas (4H) and a sodium boron compound (NaB) remain.

In this reaction, so much energy is released that the water splits into hydrogen gas (4H per 2 molecules) and oxygen (2O per 2 molecules). The oxygen thus released then bonds with the sodium boron compound, yielding sodium metaborate ($NaBO_2$) and hydrogen gas (4H). Thus, overall, four hydrogen atoms (4H) are released per molecule of sodium borohydride and 4 hydrogen atoms (4H) are released per 2 molecules of water, yielding a total of 8 hydrogen atoms (8H) and a reaction heat of 30MJ that is cooled to 90°C.

$NaBH_4 + 2H_2O \rightarrow NaBO_2 + H_2O + 8H$ + 90°C of heat.

The hydrogen released has now become hydrogen gas and, with the help of a fuel cell for generating electricity, can be used as a direct energy source; in addition to the use of the heat from the reaction, the hydrogen can be converted into heat using a catalyst.

Packing process

The residual substances, consisting of sodium metaborate (borax) and water, are removed from the mixing chamber, after which a portion of the water is evaporated. The oxygen that is bonded to the sodium boron compound is removed and; in turn, hydrogen (4H) is again affixed to it, again yielding sodium borohydride ($NaBH_4$), and the process repeats.

The hydrogen required for this is obtained by having the unpacking process take place two times simultaneously: the internal process and the external process. Both processes require sustainable electrical energy. Further, the unpacking process results in a yield of 8H.

The internal process yields 8H and in turn splits it into 2x 4H, i.e., 4H for the formation of the sodium borohydride ($NaBH_4$) needed for the repetition of its own process and 4H for the creation of sodium borohydride ($NaBH_4$) in the external process destined for market consumption. There, the 4H which has been bonded to the sodium boron compound through the splitting of the water is again converted into 8H (unpacking process).

Fundamentals 5 Code Library

maXbox

Author: Max Kleiner

"Love comes unseen; we only see it go."
– Henry Austin Dobson

INTRODUCTION
The Fundamentals 5 Code Library is a big toolbox for Delphi and FreePascal.

What I appreciate most in this library are the main utilities for network and internet. These utilities (Utils) it provides, involve math, statistic, unicode routines and data structures for classes similarly to how users would find them in a big framework.
In this way, testing-routines help ensure your tests and give you confidence in your code.

Our Test Directory includes detailed information, guides and references for many of our tests.
This includes test and result codes, specimen collection requirements, specimen transport considerations, and methodology.
Concerning a documentation of the Fundamentals library, the most is detailed direct in code with a revision history and supported compilers.
There are also tools like **DiPasDoc** which we can use to generate API documentation from comments in sources.
Those are free and generate HTML as well as CHM. An online documentary has been built in the meantime but not finished:
`http://fundamentals5.kouraklis.com/`

David J. Butler is also the author of the Zlib version of PASZLIB which is based on the zlib 1.1.2, a general purpose data compression library. The 'zlib' compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data. This version of the library supports only one compression method (deflation) but other algorithms will be added later and will have the same stream interface.

Similarly to for example the **Indy** or **Jedi** library, we can specify the class name, test name number of samples (measurements) to take and number of operations (iterations) the code will be executed.
Most of the operations are not overload but has a strong name like that:

```
function GetEnvironmentVariableA(
        const Name: AnsiString): AnsiString;
function GetEnvironmentVariableU(
        const Name: UnicodeString): UnicodeString;
function GetEnvironmentVariable(
        const Name: String): String;
{$IFDEF UseInline}inline;{$ENDIF}
```

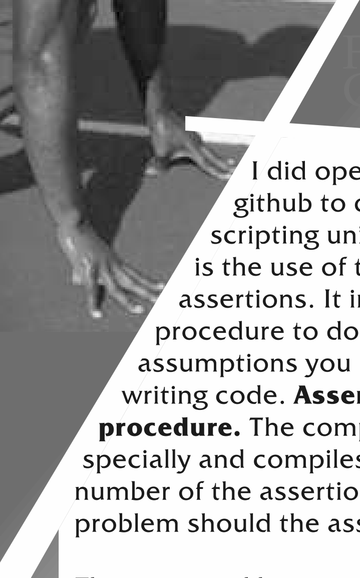
What's nice is when you pass an empty name or some invalid thing as the actual parameter of samples, most of the names or buffers will be initialized and checked or filled out with a proper default name of their own.
So the **Fundamentals Library** includes:

- String, DateTime and dynamic array routines
- Unicode routines
- Hash (e.g. SHA256, SHA512, SHA1, SHA256, MD5)
- Integer (e.g. Word128, Word256, Int128, Int256)
- Huge Word, Huge Integer
- Decimal (Decimal32, Decimal64, Decimal128,
- HugeDecimal and signed decimals)
- Random number generators
- Ciphers (symmetric: AES, DES, RC2, RC4; asymmetric: RSA, Diffie-Hellman)
- Data structures (array and dictionary classes)
- Mathematics (Rational number, complex number, vector, matrix, statistics)
- JSON parser
- Google protocol buffer parser, utilities and Pascal code generator
- Socket library (cross platform - Windows and Linux) TLS Client, TLS Server
- TCP Client, TCP Server, HTTP Client, HTTP Server, HTML Parser and XML Parser.

`https://github.com/fundamentalslib/fundamentals5/`

`https://github.com/maxkleiner/fundamentals5`

Assertions check for programming errors, NOT user errors!



I did open another fork on github to document my adapted scripting units. Another advantage is the use of test-procedure with assertions. It implements the Assert procedure to document and enforce the assumptions you must make when writing code. **Assert is not a real procedure.** The compiler handles Assert specially and compiles the filename and line number of the assertion to help you locate the problem should the assertion fail.

The syntax is like:

```
procedure Assert(Test: Boolean);
procedure Assert(Test: Boolean; const Message: string);
```

If you write a simple script program and distribute it to each computer, you can have the users start the tests on their own by running the script with a list of asserts.

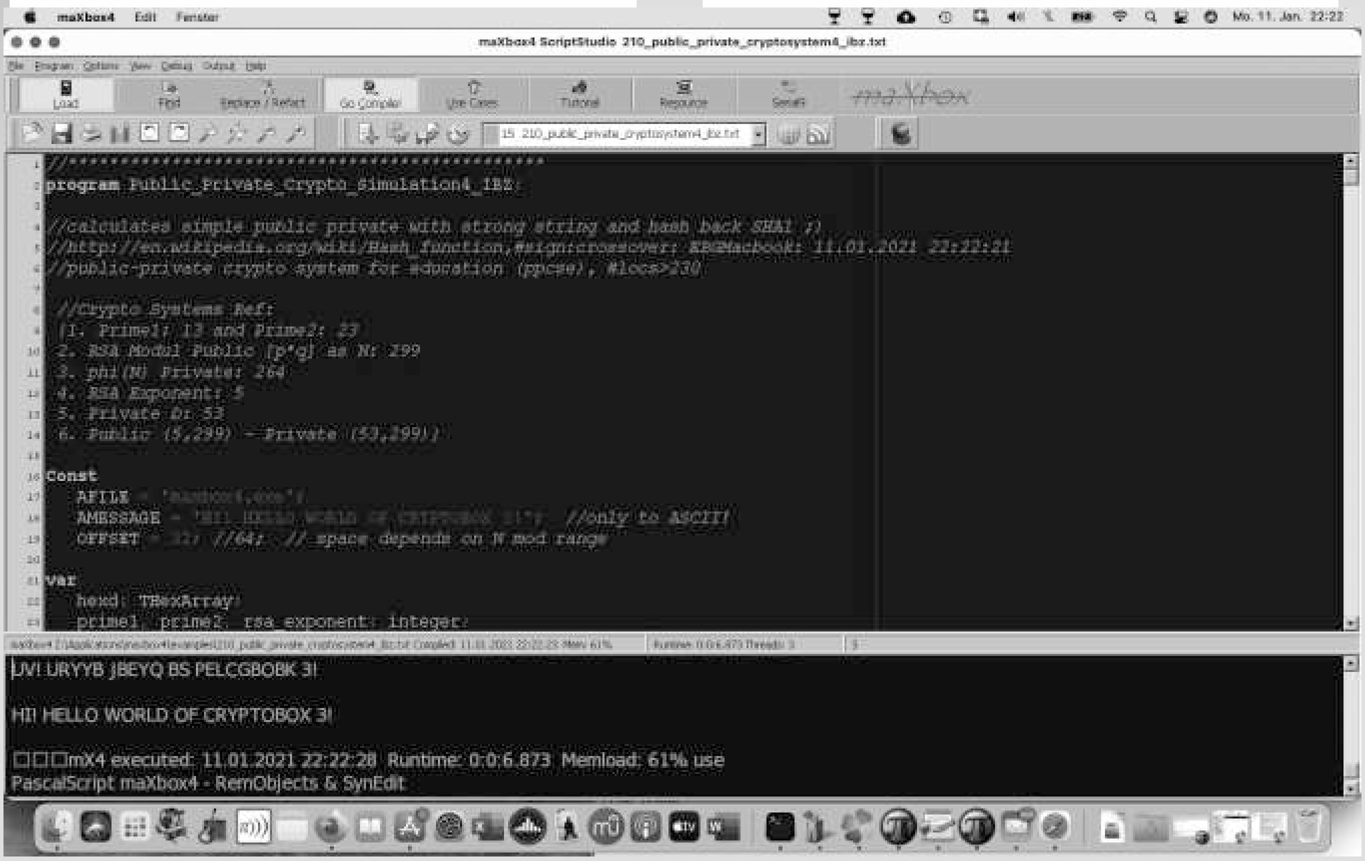
```
Assert(CopyFrom('a', 0) = 'a', 'CopyFrom');
Assert(CopyFrom('a', -1) = 'a', 'CopyFrom');
Assert(CopyFrom('', 1) = '', 'CopyFrom');
Assert(CopyFrom('', -2) = '', 'CopyFrom');
Assert(CopyFrom('1234567890', 8) = '890', 'CopyFrom');
Assert(CopyFrom('1234567890', 11) = '', 'CopyFrom');
Assert(CopyFrom('1234567890', 0) = '1234567890', 'CopyFrom');
Assert(CopyFrom('1234567890', -2) = '1234567890', 'CopyFrom');

Assert(not StrMatch('', '', 1), 'StrMatch');
Assert(not StrMatch('', 'a', 1), 'StrMatch');
Assert(not StrMatch('a', '', 1), 'StrMatch');
Assert(not StrMatch('a', 'A', 1), 'StrMatch');
Assert(StrMatch('A', 'A', 1), 'StrMatch');
Assert(not StrMatch('abcdef', 'xx', 1), 'StrMatch');
Assert(StrMatch('xbcdef', 'x', 1), 'StrMatch');
Assert(StrMatch('abcdxxxxx', 'xxxxx', 5), 'StrMatch');
Assert(StrMatch('abcdef', 'abcdef', 1), 'StrMatch');
Assert(StrMatch('abcde', 'abcd', 1), 'StrMatch');
Assert(StrMatch('abcde', 'abc', 1), 'StrMatch');
Assert(StrMatch('abcde', 'ab', 1), 'StrMatch');
Assert(StrMatch('abcde', 'a', 1), 'StrMatch');
Assert(StrMatches('abcd', 'abcd', 1)=true, 'StrMatches');
```

Lets take the above single assert with

```
Function StrMatches(const Substr, S: AnsiString; const Index: Int): Boolean;
```

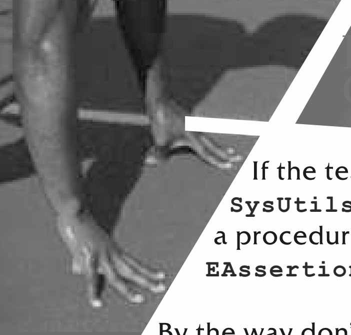
As you can see the strings matches if equal otherwise we get an Exception:

```
Assert(StrMatches('abcd', 'abcde', 1)=true, 'StrMatches');
```
Exception: StrMatches


maXbox4 Edit Fenster
Mo. 11. Jan. 22:22
maXbox4 ScriptStudio 210_public_private_cryptosystem4_ibz.txt
File Program Options View Debug Output Help
Load
Find
Replace / Refact
Go Compiler
Use Cases
Tutorial
Resource
Search
15 210_public_private_cryptosystem4_ibz.txt

```
//**************************************************
program Public_Private_Crypto_Simulation4_IBZ;

//calculates simple public private with strong string and hash back SHA1 ;)
//http://en.wikipedia.org/wiki/Hash_function,#signtcrossover: EB2Macbook: 11.01.2021 22:22:21
//public-private crypto system for education (ppcse), #locs>230

 //Crypto Systems Ref:
 {1. Prime1: 13 and Prime2: 23
 2. RSA Modul Public [p*q] as N: 299
 3. phi(N) Private: 264
 4. RSA Exponent: 5
 5. Private D: 53
 6. Public (5,299) - Private (53,299)}

Const
   AFILE = 'maXbox4.exe';
   AMESSAGE = 'HI! HELLO WORLD OF CRYPTOBOX 3!'; //only to ASCII!
   OFFSET = 32; //64; // space depends on N mod range

var
   hexd: THexArray;
   prime1, prime2, rsa_exponent: integer;
```

```
UV! URYYB JBEYQ BS PELCGBOBK 3!

HI! HELLO WORLD OF CRYPTOBOX 3!

□□□mX4 executed: 11.01.2021 22:22:28 Runtime: 0:0:6.873 Memload: 61% use
PascalScript maXbox4 - RemObjects & SynEdit
```



If the test condition fails the **SysUtils** unit sets this variable to a procedure that raises the **EAssertionFailed exception**.

By the way don't comment an assert like this:

```
//Assert(StrMatchLeft('ABC1D', 'aBc1', False), 'StrMatchLeft');
//Assert(StrMatchLeft('aBc1D', 'aBc1', True), 'StrMatchLeft');
```

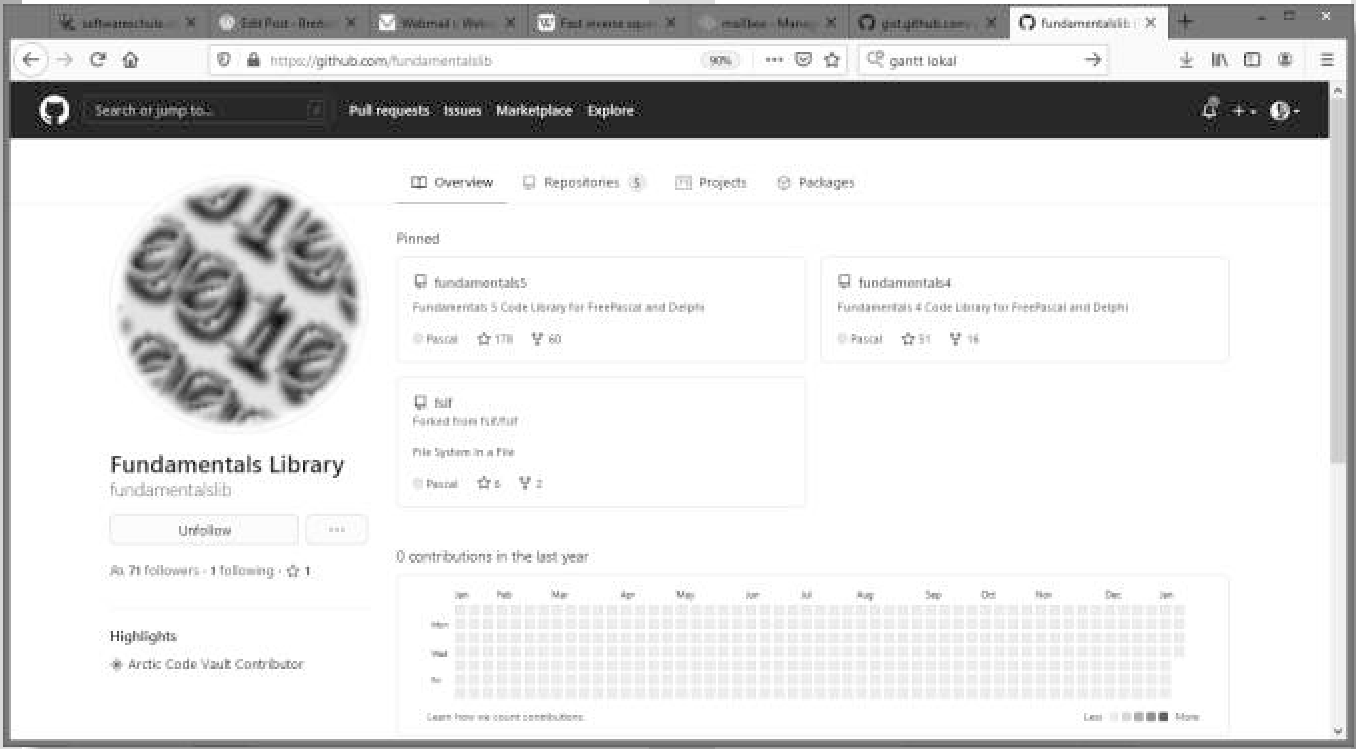
You also can negate an assert as long as it delivers a **boolean** (logic) condition:

```
 Assert(not StrMatchLeft('AB1D', 'ABc1', False), 'StrMatchLeft');
Assert(not StrMatchLeft('aBC1D', 'aBc1', True), 'StrMatchLeft');
```

Then you want to write more assert system information to a log file for analyzing problems during installation, debugging, tests and de-installation or app distribution like that:

```
10/01/2018 19:31:54 V:4.6.2.10
[max] problem occurred in initializing MCI.
[at:  3275216pgf; mem:1247492]
14/01/2018 17:15:18 V:4.7.2.30
[max] MAXBOX8 Out Of Range.
[at:  2607048pgf; mem:1082444]
14/01/2018 17:15:21 V:4.7.2.40
[max] MAXBOX8 Out Of Range.
[at:  2605716pgf; mem:1080012]
16/01/2018 09:18:00 V:4.7.5.20
[max] MAXBOX8 List index out of bounds
(456). [at:  2913700pgf; mem:1157700]
```

```
{                                        }
{ Test cases                             }
{                                        }
{$IFDEF DEBUG}
{$IFDEF LOG}
{$IFDEF TEST}
//{$ASSERTIONS ON}
```



Fundamentals Library
fundamentalslib

71 followers · 1 following · 1

Highlights
Arctic Code Vault Contributor

fundamentals5
Fundamentals 5 Code Library for FreePascal and Delphi
Pascal 178 60

fundamentals4
Fundamentals 4 Code Library for FreePascal and Delphi
Pascal 51 16

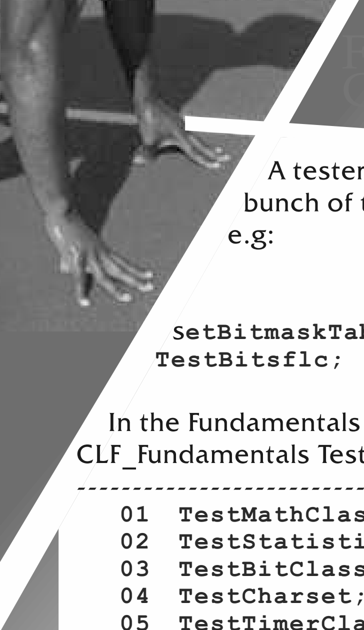
fsif
Forked from fsif/fsif
File System in a File
Pascal 6 2

0 contributions in the last year

Learn how we count contributions.

Next step is to bundle asserts in a
Test Procedure with sections like that:

```
procedure TestBitsflc;
begin
 Assert(SetBit32($100F, 5) = $102F,          'SetBit');
 Assert(ClearBit32($102F, 5) = $100F,        'ClearBit');
 Assert(ToggleBit32($102F, 5) = $100F,       'ToggleBit');
 Assert(ToggleBit32($100F, 5) = $102F,       'ToggleBit');
 Assert(IsBitSet32($102F, 5),                'IsBitSet');
 Assert(not IsBitSet32($100F, 5),            'IsBitSet');
 Assert(IsHighBitSet32($80000000),           'IsHighBitSet');
 Assert(not IsHighBitSet32($00000001),       'IsHighBitSet');
 Assert(not IsHighBitSet32($7FFFFFFF),       'IsHighBitSet');

 Assert(SetBitScanForward32(0) = -1,         'SetBitScanForward');
 Assert(SetBitScanForward32($1020) = 5,      'SetBitScanForward');
 Assert(SetBitScanReverse32($1020) = 12,     'SetBitScanForward');
 Assert(SetBitScanForward321($1020, 6) = 12,   'SetBitScanForward');
 Assert(SetBitScanReverse321($1020, 11) = 5,   'SetBitScanForward');
 Assert(ClearBitScanForward32($FFFFFFFF) = -1, 'ClearBitScanForward');
 Assert(ClearBitScanForward32($1020) = 0,     'ClearBitScanForward');
 Assert(ClearBitScanReverse32($1020) = 31,    'ClearBitScanForward');
 Assert(ClearBitScanForward321($1020, 5) = 6,  'ClearBitScanForward');
 Assert(ClearBitScanReverse321($1020, 12) = 11, 'ClearBitScanForward');

 Assert(ReverseBits32($12345678) = $1E6A2C48,  'ReverseBits');
 Assert(ReverseBits32($1) = $80000000,        'ReverseBits');
 Assert(ReverseBits32($80000000) = $1,        'ReverseBits');
 Assert(SwapEndian32($12345678) = $78563412,   'SwapEndian');

 Assert(RotateLeftBits32(0, 1) = 0,          'RotateLeftBits32');
 Assert(RotateLeftBits32(1, 0) = 1,          'RotateLeftBits32');
 Assert(RotateLeftBits32(1, 1) = 2,          'RotateLeftBits32');
 Assert(RotateLeftBits32($80000000, 1) = 1,  'RotateLeftBits32');
 Assert(RotateLeftBits32($80000001, 1) = 3,  'RotateLeftBits32');
 Assert(RotateLeftBits32(1, 2) = 4,          'RotateLeftBits32');
 Assert(RotateLeftBits32(1, 31) = $80000000, 'RotateLeftBits32');
 Assert(RotateLeftBits32(5, 2) = 20,         'RotateLeftBits32');
 Assert(RotateRightBits32(0, 1) = 0,         'RotateRightBits32');
 Assert(RotateRightBits32(1, 0) = 1,         'RotateRightBits32');
 Assert(RotateRightBits32(1, 1) = $80000000, 'RotateRightBits32');
 Assert(RotateRightBits32(2, 1) = 1,         'RotateRightBits32');
 Assert(RotateRightBits32(4, 2) = 1,         'RotateRightBits32');

 Assert(LowBitMask32(10) = $3FF,             'LowBitMask');
 Assert(HighBitMask32(28) = $F0000000,       'HighBitMask');
 Assert(RangeBitMask32(2, 6) = $7C,          'RangeBitMask');

 Assert(SetBitRange32($101, 2, 6) = $17D,     'SetBitRange');
 Assert(ClearBitRange32($17D, 2, 6) = $101,   'ClearBitRange');
 Assert(ToggleBitRange32($17D, 2, 6) = $101,   'ToggleBitRange');
 Assert(IsBitRangeSet32($17D, 2, 6),          'IsBitRangeSet');
 Assert(not IsBitRangeSet32($101, 2, 6),      'IsBitRangeSet');
 Assert(not IsBitRangeClear32($17D, 2, 6),    'IsBitRangeClear');
 Assert(IsBitRangeClear32($101, 2, 6),        'IsBitRangeClear');
 Assert(IsBitRangeClear32($101, 2, 7),        'IsBitRangeClear');
end;
{$ENDIF}
{$ENDIF}
```



A tester is then able to run a bunch of tests in Fundamentals, e.g:

```
  SetBitmaskTable;
 TestBitsflc;
```

In the Fundamentals Lib we do have a 15 CLF_Fundamentals Testroutines Package:

```
-----------------------------------
   01  TestMathClass;
   02  TestStatisticClass;
   03  TestBitClass;
   04  TestCharset;
   05  TestTimerClass
   06  TestRationalClass
   07  TestComplexClass
   08  TestMatrixClass;
   09  TestStringBuilderClass
   10  TestASCII;
   11  TestASCIIRoutines;
   12  TestPatternmatcher;
   13  TestUnicodeChar;
   14  flcTest_HashGeneral;
   15  flcTest_StdTypes;
```

```
procedure TestStdTypes;
begin
  {$IFDEF LongWordIs32Bits}    Assert(SizeOf(LongWord)    = 4);  {$ENDIF}
  {$IFDEF LongIntIs32Bits}     Assert(SizeOf(LongInt)     = 4);  {$ENDIF}
  {$IFDEF LongWordIs64Bits}    Assert(SizeOf(LongWord)    = 8);  {$ENDIF}
  {$IFDEF LongIntIs64Bits}     Assert(SizeOf(LongInt)     = 8);  {$ENDIF}
  {$IFDEF NativeIntIs32Bits}   Assert(SizeOf(NativeInt)   = 4);  {$ENDIF}
  {$IFDEF NativeIntIs64Bits}   Assert(SizeOf(NativeInt)   = 8);  {$ENDIF}
  {$IFDEF NativeUIntIs32Bits}  Assert(SizeOf(NativeUInt)  = 4);  {$ENDIF}
  {$IFDEF NativeUIntIs64Bits}  Assert(SizeOf(NativeUInt)  = 8);  {$ENDIF}
end;
```

Another way is to prevent call errors as a mistaken precondition of false assumption in a procedure you designed. This pre- and postcondition can handle a lot of errors.
An example should make this clear.
A `TStack` object has a method called `Pop` to remove the topmost data object from the stack.
If the `stack` is empty, I count calling `Pop` as a programming mistake: you really should check for the stack being empty in your program prior to calling `Pop`. Of course `Pop` could have an if statement within it that did this check for you, but in the ***MAJORITY*** of cases the stack won't be empty when `Pop` is called and in the ***MAJORITY*** of cases when you use Pop, you will have some kind of loop in your program which is continually checking whether the stack is empty or not anyway. In my mind having a check for an empty stack within `Pop` is safe but slow.
So, instead, `Pop` has a call to an `Assert procedure` at the start (activated by the DEBUG compiler define) that checks to see whether the stack is empty. Here is the code for `Pop`:

```
function TStack.Pop : pointer;
var
 Node : PNode;
begin
 {$IFDEF DEBUG}
  Assert(not IsEmpty, ascEmptyPop);
 {$ENDIF}
 Node := Head^.Link;
 Head^.Link := Node^.Link;
 Pop := Node^.Data;
 acDisposeNode(Node);
end;
```

As you see, if DEBUG is set the Assert procedure checks whether the stack is empty first, if not it executes the code that pops the data object off the stack.
If the stack is empty an EEZAssertionError exception is raised (the `constant ascEmptyPop` is a string code for a string-table resource).
If DEBUG is not set the code runs at full speed.

So log the steps and compare test procedures before installation:
The location of the update can be a local, UNC or network path to compare it.
If you need Admin Rights you can try this:

```
ExecuteShell('cmd','/c runas
"/user:Administrator" '+
ExePath+'maXbox4.exe')
```
or
```
C:> net user Administrator /active:yes
```

After you have the option activated hereby of finishing and writing the script, the next and final step is select `"Go Compile"` in maXbox.
What this does is create a complete, ready-to-run Setup program based on your script.
By default, this is created in a directory named Exepath under the directory or UNC path containing the script or what destination you need.

```
function GetInstallScript(const S_API, pData:
string): string;
var ts: TStrings;
begin
  with TIdHTTP.create(self) do begin
    try
     ts:= TStringList.Create
     ts.Add('install='+HTTPEncode(pData));
     result:= Post(S_API,ts);
    finally
     ts.Free;
     Free;
    end;
  end
end;
```

The most important step comes with unit tests with setup and teardown.
Generic "Assert This" Assertion Procedure means that most generic assertion program simply says "assert this" and passes a Boolean expression.
It is used by all the other assertion routines, which construct a Boolean expression from their specific values and logic.
Unit testing is a way of testing the smallest item of code referred to as a unit that can be logically isolated in a system.

There are different unit (modules) test frameworks for **Delphi** and **Free Pascal**, which can cause duplicate work for those who target both compilers (for example, library and framework developers).
Default unit test framework for Free Pascal is FPCUnit, it has almost the same design as DUnit but different in minor details.

A unit can be almost anything you want it to be – a specific piece of functionality, a program, or a particular method within the application:

```
type
 THugeCardinal_TestCase = TTestCase;
var
  Fbig1234: THugeCardinal;
  Fbig2313: THugeCardinal;
  Fbig3547: THugeCardinal;
  //TVerifyResult
  Temp1, Temp2, Temp3, Temp4: THugeCardinal;
  Temp2000_1: THugeCardinal;
  Temp2000_2: THugeCardinal;
  T3, F100: THugeCardinal;
  TmpStream: TMemoryStream;

  procedure THugeCardinal_TestCaseSetUp; //override;
  procedure THugeCardinal_TestCaseTearDown; //override;

 //published
    //procedure Test_CreateZero;
    procedure Test_CreateRandom;
    procedure Test_CreateSmall;
    procedure Test_Clone;
    procedure Test_Assign;
    procedure Test_Zeroise;
    procedure Test_CompareSmall;
    procedure Test_Compare;
    procedure Test_AssignSmall;
    procedure Test_BitLength;
    procedure Test_MaxBits;
    procedure Test_Add;
    procedure Test_Increment;
    procedure Test_Subtract;
    procedure Test_MulPower2;
    procedure Test_MulSmall;
    procedure Test_Multiply;
    procedure Test_Modulo;
    procedure Test_AddMod;
    procedure Test_MultiplyMod;
    procedure Test_isOdd;
    procedure Test_CreateFromStreamIn;
    procedure Test_CloneSized;
    procedure Test_Resize;
    procedure Test_AssignFromStreamIn;
    procedure Test_Swap;
    procedure Test_ExtactSmall;
    procedure Test_StreamOut;
    procedure Test_PowerMod;
    procedure Test_SmallExponent_PowerMod;

  procedure InitUnit_HugeCardinalTestCases;
  begin
  //TestFramework.RegisterTest( THugeCardinal_TestCase.Suite)
    THugeCardinal_TestCaseSetUp;
  end;

  procedure DoneUnit_HugeCardinalTestCases;
  begin
   THugeCardinal_TestCaseTearDown
  end;
```

CONCLUSION:
The proper way to use **Assert** in the **Fundamentals Lib** is to specify conditions that must be true in order for your code to work correctly.

```
 Assert(StrMatches('abcd', 'abcde', 1)=true, 'StrMatches');
```

All programmers make assumptions about internal state of an object or function, the value or validity of a subroutine's arguments, or the value returned from a function. A good way to think about assertions is that they check for programmer errors, not user errors!

My 7 Steps for maintainable code:
- Maintain separation of concerns (avoid unnecessary dependencies)
- Fully qualified unit names to be used: Winapi.Windows not Windows
- Code format to be consistent with LIB source
- Do not put application-specific implementations in general code libraries
- Carefully consider modification to common code – the way to proceed
- No hints (instant code review fail) and No warnings
- Keep code small – avoid long methods and should be broken down

Ref:

`http://www.softwareschule.ch/download/maxbox_starter36.pdf`
`https://github.com/fundamentalslib/fundamentals5/`
`http://www.softwareschule.ch/examples/unittests.txt`
script: 919_uLockBox_HugeCardinalTestCases.pas

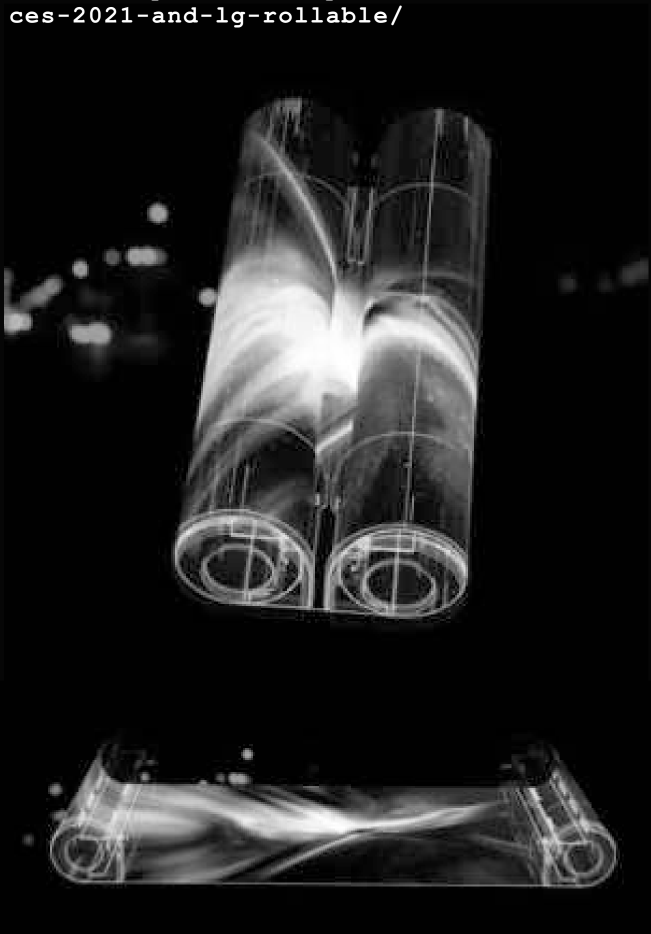
Doc:
`http://fundamentals5.kouraklis.com/`
`https://maxbox4.wordpress.com`

News from the future

About 5 years ago I talked about future development . One of the items was a rollable screen from an item like a pencil.

In addition to the rollable phone, there was announced a new 17-inch printed OLED scrolling display that can be unfurled and features a "100% color gamut. The new screen technology, from TCL CSOT, can be widely applied on flexible TVs, curved and foldable displays as well as transparent commercial display screens.

`https://www.cnet.com/news/tcl-rollable-phone-concept-unveiled-at-ces-2021-and-lg-rollable/`



LAZARUS: CREATING ATOM AND VS CODE PLUGINS IN PASCAL

By Michael van Canneyt

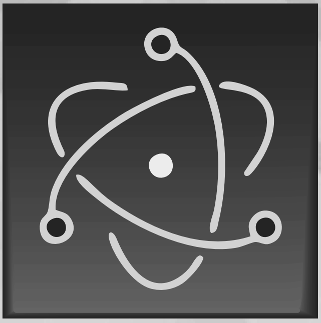


Figure 1: Logo of Electron

ABSTRACT
The Atom and VisualStudio Code editors are among the most popular programmer editors. These editor are extensible for anyone that can create Javascript. **Object Pascal programmers can also create Javascript, so logically they can also create VS Code and Atom extensions.**
In this article we show how.

Electron (formerly known as Atom Shell) is an open-source software framework developed and maintained by GitHub. It allows for the development of desktop GUI applications using web technologies: it combines the Chromium rendering engine and the Node.js runtime. Electron is the main GUI framework behind several open-source projects including Atom, GitHub Desktop, Light Table, Visual Studio Code, Evernote, and WordPress Desktop.

Atom is a free and open-source text and source code editor for macOS, Linux, and Microsoft Windows with support for plug-ins written in JavaScript, and embedded Git Control, developed by GitHub. Atom is a desktop application built using web technologies. Most of the extending packages have free software licenses and are community-built and maintained. Atom is based on Electron (formerly known as Atom Shell), a framework that enables cross-platform desktop applications using Chromium and Node.js. It is written in CoffeeScript and Less.

Atom was released from beta, as version 1.0, on 25 June 2015. Its developers call it a "hackable text editor for the 21st Century". It is fully customizable in HTML, CSS, and JavaScript.

LESS KNOWN IS THAT YOU CAN RUN WEB APPLICATIONS ON A DESKTOP:

1 INTRODUCTION

Web applications are cross-platform. Any platform that has a browser can run a web application.

Less known is that you can run web applications on a desktop:
Electron is an environment that uses the browser and **Node.js** to allow you to create desktop applications that are written in Javascript, and which run in a browser - sort of. It uses the chromium engine to render HTML - the HTML is the GUI (Graphical User Interface) of the application.
The application logic is written in Javascript.

With the appearance of Electron, two powerful programming editors were created that use Electron: Atom and VS Code.
Since they are built with Electron, they are cross-platform.

Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. Microsoft has released Visual Studio Code's source code on the VSCode repository of GitHub, under the permissive MIT License, while the compiled releases are freeware.

Her you can download the **Atom** program:
`https://atom.io/`
Here you can download **VS Code** programm
`https://code.visualstudio.com/download`

With the appearance of **Electron,** two powerful programming editors were created that use **Electron: Atom** and **VS Code**.
Since they are built with Electron, they are cross-platform. And because Electron uses a Javascript engine (the same as Node.js), it is possible to loadand execute arbitrary Javascript code in the editor. This Electron ability is used to allow people to extend the editor: you can extend the editor in whatever way you see fit. The only requirement is that the extension is written in Javascript.

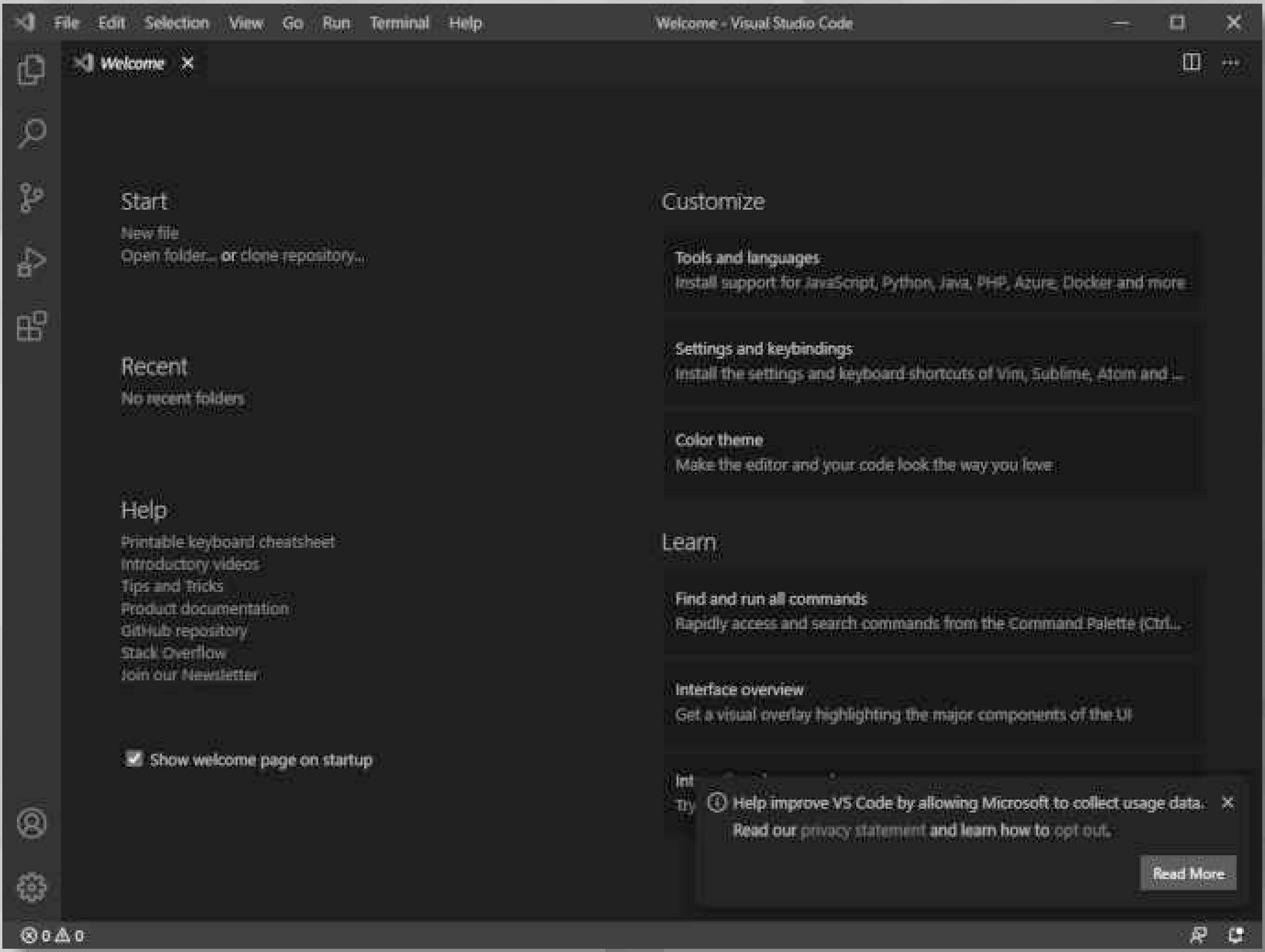

File Edit Selection View Go Run Terminal Help
Welcome - Visual Studio Code
Welcome
Start
New file
Open folder... or clone repository...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, Python, Java, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and ...
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl...
Interface overview
Get a visual overlay highlighting the major components of the UI
Help improve VS Code by allowing Microsoft to collect usage data.
Read our privacy statement and learn how to opt out.
Read More
0 0

Figure 2: Visual Studio Code Editor

Naturally, there are numerous plugins for both editors: in Atom they are called Packages, in VS Code, they are called Extensions. Both editors have plugins to facilitate programming in Object Pascal.
But can you also write pascal to extend these editors ?
Fortunately, now you can.
Pas2JS and the TMS Web Compile convert Object Pascal to Javascript.
Javascript is a simple text file, and it should be possible to include the output of the Pas2JS or TMS Web Compiler in the editor in a format it accepts.
Note that the Lazarus IDE support requires the trunk version of the Pas2JS compiler: it uses a `-ja` option to append a small piece of Javascript to the output. **If you use a released version of the compiler, you can append this little piece of code manually.**

2 EDITOR API

The Atom and VS Code editors are themselves written in Javascript. Because they are well-designed, they make an API available to any Javascript programmer that wishes to extend the editor. How can we make this API available to the pascal programmer ? In the exact same manner as the Browser API was made available to the Pascal programmer: by writing external class definitions that 'translate' the Javascript APIs of the editors for the Object Pascal compiler.

This task has been accomplished: the APIs of both editors have been translated to Pascal: the units that contain these definitions have been created and made available in the subversion repository. The units with the editor APIs are called **libatom** and **libvscode.**

The APIs made available to you by these editors are huge. They contain hundreds of classes. They match the browser APIs for size, so needless to say that an in-depth study of these APIs is outside the scope of this article. Although they offer the same kind of functionalities, the APIs of both editors are wildly different: code designed to run in 1 editor will not run in the other. Conceivably, a kind of unifying API can be made on top of these APIS, so as to allow a programmer to create a plugin that works for both editors.

3 PLUGIN ARCHITECTURE

Both editors have more or less the same architecture for a plugin: a plugin is similar to a library: it is a javascript file that must expose a number of functions, plus a manifest file describing the plugin. The manifest file is a JSON file such as it is found in many Node.js packages: package. json, which must have some entries for the Editor to be able to load your plugin: the location of the Javascript file (a module) with the plugin code.

In the case of VS Code, the javascript code must export two procedures:

❶ **`activate`**

This function is called when the plugin is loaded: In this function, you must install the necessary commands, hooks and keyboard shortcuts. For this purpose, the editor passes a context object as the sole argument to the procedure. The context contains an instance of the global editor object.

❷ **`deactivate`**

This function is called when the plugin is unloaded.

In the case of Atom, the javascript code must export the above two functions (although they have different arguments), and can export two additional procedures:

❸ **`initialize`**

This function is called exactly once before activating the package.

❹ **`serialize`**

This function is called when the package is being unloaded, so it can preserve state. The saved state is passed to the package activate function when the editor loads it.

The module concept of javascript translates almost directly to Libraries.
Support for libraries is currently being implemented, but is not yet in the current release.

Because the **Pas2JS compiler (2.0)** does not yet support compiling libraries, it is easiest to use a little Javascript wrapper that will load the code generated by Pas2js. This little Javascript wrapper will start the pas2js rtl, and will call a function defined in the main program. For Atom, this wrapper looks like this:

```
'use babel';
import { CompositeDisposable } from 'atom';
import { pas, rtl } from './pas2jsdemopackage.js';
export default {
   activate(state) {
      rtl.run();
      this.subscriptions = new CompositeDisposable();
      this.atomEnv = {
         atomGlobal : atom,
         subscriptions : this.subscriptions,
         initialState : state
      }
      this.atomHandler = {
         onDeactivate : function (a) {},
         onSerialize : function (a,o) {}
      }
      pas.program.InitAtom(this.atomEnv,this.atomHandler);
   },

   deactivate() {
      if (this.atomHandler.onDeactivate) {
         this.atomHandler.onDeactivate(this.atomEnv)
      }
   this.subscriptions.dispose();
   },

   serialize() {
      var obj = {};
      if (this.atomHandler.onSerialize) {
      this.atomHandler.onSerialize(this.atomEnv,obj)
      }
      return obj;
   }
};
```

You don’t need to be a Javascript specialist to understand that this code exports the three functions mentioned above. What is important for the pascal programmer, are three lines of code.
The first important line imports the symbols that can be found in any pas2js generated program:

```
import { pas, rtl } from './pas2jsdemopackage.js';
```

The pas object contains the code from all the units and the main program. The rtl object contains the pascal run-time code.
The second important line of code initializes the pascal runtime:
`rtl.run();`
This is the same code that can be found in a HTML page `<script>` tag to start a pascal-generated program.
The last piece of code transfers control to the InitAtom function in the pascal main program:

```
pas.program.InitAtom(this.atomEnv,this.atomHandler);
```

This last line is a design choice, a convention to handle the transfer of code to pascal;
it is only necessary for the time being, because Pas2JS does not yet support libraries.

When Pas2JS will support libraries, the above wrapper will of course no longer be necessary.

Note that a set of callbacks is passed to the `InitAtom` call. This is done so only one function needs to be exposed: when the `InitAtom` function returns, the handlers in the `atomHandler` object will be set and can be used to transfer control to the pascal code in the other two exposed functions.
The VS Code wrapper is entirely similar, except the name of the Pascal initialization function:

```
const vscode = require('vscode');
const pascalRuntime = require('./pas2jsdemoextension.js');
var callbacks = {
     onDeactivate: function (a) { }
}
function activate(context) {
     pascalRuntime.rtl.run();
     var vscodeEnv = {
        vscodeGlobal: vscode,
        extensionContext: context
     }
   pascalRuntime.pas.program.InitVSCode(vscodeEnv,callbacks);
}
function deactivate() {
   if (callbacks.onDeactivate) {
     callbacks.onDeactivate();
   }
}
   module.exports = {
     activate,
     deactivate
}
```

4 OBJECT PASCAL APPLICATION

In the above, we've seen the Javascript code that will be used to kickstart the pascal code for our plugin.

To make it easier for you to create an Object Pascal program that can be used as a VS Code or Atom plugin, a unit was created as part of the Pas2JS package, which contains an 'Application' object.

The application object is much like the Delphi `TApplication` class, as it descends from `TCustomApplication` - a standard class in the Free Pascal runtime, which serves as the ancestor for all kinds of application classes - native or browser-based:
console applications, GUI applications (in Lazarus) and Node.js or Browser based applications in Pas2JS.

Because the VS Code and Atom APIs are different, the application objects for both environments
are of course also different. So the Pas2JS distribution now has two units called **atomapp** and **vscodeapp.**
They each define an application object for use in the Atom and VS Code editor: Each object has a property that contains an instance of the global VSCode and Atom editor environment: these objects are made available by the editors: we'll see how to use this later on.

```
TAtomApplication = class(TCustomApplication)
  Protected
    procedure DoActivate(aState : TJSObject); virtual;
    procedure DoDeactivate(); virtual;
    procedure DoSerialize(aState : TJSObject); virtual;
  Public
    procedure SaveAtomEnvironment(aEnv : TAtomEnvironment;
                                  aCallBacks : TAtomPackageCallBacks);
    property Subscriptions : TAtomCompositeDisposable;
    Property Atom : TAtom;
end;
```

This is a simple application object (although some methods have been left out for clarity). It exposes two Javascript objects: **`SubScriptions`** and **`Atom`**, which will contain the Atom global editor object that exposes the complete Atom API for you, and a subscriptions object.
The subscriptions object is an Atom object that owns all resources you will allocate during the lifetime of your plugin. When the plugin is freed, the subscription will free all objects it owns.

To create your Atom plugin, the package program code must define a descendent of this class, and override the three protected virtual methods

```
TMyAtomApplication = Class(TAtomApplication)
  Protected
    procedure DoActivate(aState : TJSObject); override;
    procedure DoDeactivate; override;
    procedure DoSerialize(aState : TJSObject); override;
end;
```

These methods obviously correspond to the 3 exported functions of the Javascript wrapper and will be called when the plugin is loaded and unloaded.

The Javascript wrapper code calls a function **`InitAtom`** (this method name is case sensitive). In this function, you can instantiate the atom application class, and call **`SaveAtomEnvironment`** to save the atom object and set the callbacks needed by the wrapper:

```
Procedure InitAtom(aAtom : TAtomEnvironment;
          aCallBacks : TAtomPackageCallBacks);
begin
  If Application=Nil then
    Application:=TMyAtomApplication.Create(Nil);
  Application.SaveAtomEnvironment(aAtom,aCallBacks);
end;
```

This will also call the application object's **`DoActivate`** method. The VS Code application object looks very similar:

```
TVSCodeApplication = class(TCustomApplication)
  Protected
    procedure DoActivate; virtual;
    procedure DoDeactivate(); virtual;
  Public
  procedure SaveVSCodeEnvironment(aEnv :TVSCodeEnvironment;
                       aCallBacks : TVSCodeExtensionCallBacks);
    Property VSCode : TVSCode;
    Property ExtensionContext : TVSExtensionContext;
end;
```

Likewise, a VS Code extension program must contain a descendent of this class, and it must instantiate it in the **`InitVSCode`** function that is called by the Javascript wrapper.

```
 Procedure InitVSCode(aVSCode : TVSCodeEnvironment;
  aCallBacks : TVSCodeExtensionCallBacks);
begin
  If Application=Nil then
     Application:=TMyVSCodeExtension.Create(Nil);
  Application.SaveVSCodeEnvironment(aVSCode,aCallBacks);
end;
```

Again, the **`DoActivate`** method of the application class will be called by this process. In the **`DocActivate`** method, you must place all code that will hook into the editor API: add commands, keyboard shortcuts etc.

5 LAZARUS INTEGRATION

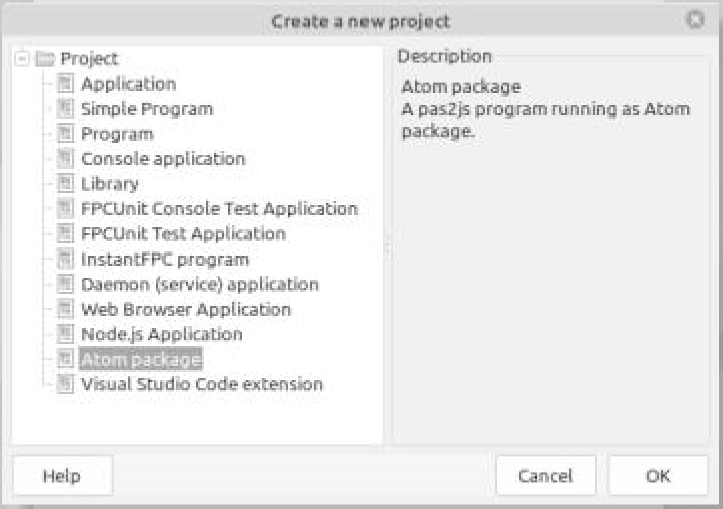
The above code is the start of a VS Code or Atom plugin:
It is possible to create an Atom or VS Code plugin using Atom or VS Code themselves to edit the pascal code, and use the above as a starting point: in that case you must manually add the wrapper code, **`package.json`** program file etc. to your project.

But at the moment of writing, the Lazarus code editor is still much more suitable for writing Object Pascal than VS Code or Atom:
The Lazarus code tools provide much more possibilities than either of these general-purpose editors do.
That is why the Lazarus Pas2Js support has been extended with two project types to create an Atom or VS Code plugin.

These wizards will create a skeleton project for you, which can be compiled using Pas2JS and which is ready to be installed in the Atom or VS Code editor. We'll demonstrate the use of these wizards in the following sections.

6 A SAMPLE ATOM PACKAGE

As a Pascal and SQL programmer, the author of this article is used not to have to care about the case of keywords and identifiers.
However, not everyone shares this view.
In Delphi and Lazarus, the code formatter can be used to remedy this sloppiness:
the IDE code formatter will happily correct casing for you.
Unfortunately, this disregard for casing extends to project documentation, SQL statements, resulting in documentation with for example SQL keywords written in a wide variety of casings.



Atom package
A pas2js program running as Atom package.

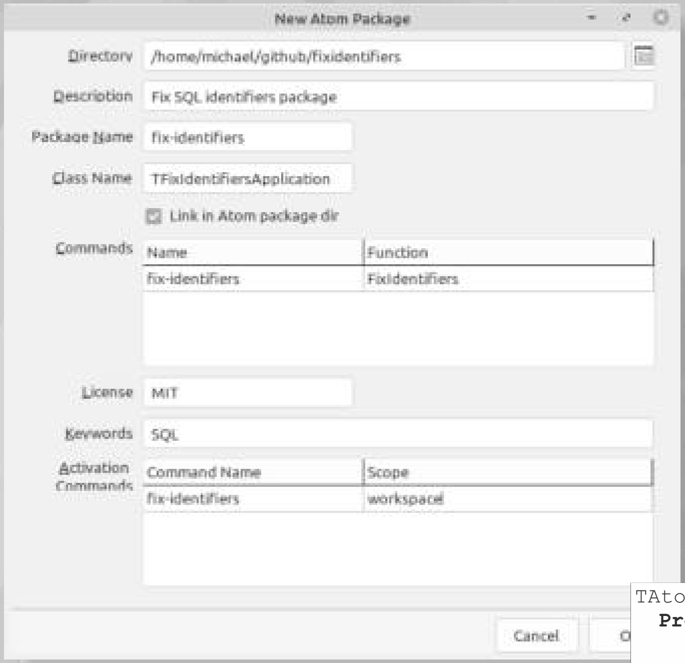
Figure 3: New project type: Atom Package

The author writes documentation primarily using Markdown, in Atom.
So, since we now can write plugins in Pascal, a good first attempt at a plugin is a plugin to fix casing of some SQL keywords:
we uppercase a selection of SQL keywords, thus having a more unified look for all SQL.

Before starting to code this, it is a good idea to look at the APIs made available to you in the Atom flight manual:
`https://flight-manual.atom.io/api/v1.54.0/AtomEnvironment/`

The Atom flight manual also contains a good anatomy of an Atom package, it describes all files that Atom expects to find. To get started, we invoke the Lazarus wizard to create an Atom plugin, as shown in figure figure 3 on page 22.
When selected, a small dialog appears to set some options, as shown in figure 4 on page 22:
The various items that can be entered here serve to generate a skeleton project:

- **Directory**
Every Atom plugin lives in its own directory. Here you specify the directory for the new plugin.
- **Description**
A textual description of your package, it goes in the manifest file (`package.json`).
- **Package Name**
A (unique) name for your package, it goes in the manifest file.
- **Class Name**
The Pascal class name for the application class.
- **Link in Atom package dir**
When you check this flag, the IDE will create (on unix and MacOS) a link in the Atom package directory to the directory where you create your project. When you next start Atom, it will then load your plugin.
- **Commands**
The commands that your package will provide to the editor. For each command
you must specify a unique name, and the name of a pascal function that will be called when the command is invoked. The command names are entered in the created menu.json file but also in the pascal code, to register the callback for the command. An empty function will be generated for each function you specify here.
- **License**
The license for your package, it goes in the manifest file.
- **Keywords** Some keywords (space separated) for your package, it goes in the manifest file.
- **Activation Commands**
The commands that will cause your package to be loaded by the editor. The scope is a valid Atom scope identifier such as atom-workspace. You can leave this list empty.


New Atom Package

Directory	/home/michael/github/fixidentifiers
Description	Fix SQL identifiers package
Package Name	fix-identifiers
Class Name	TFixIdentifiersApplication
	☑ Link in Atom package dir

Commands

Name	Function
fix-identifiers	FixIdentifiers

License	MIT
Keywords	SQL

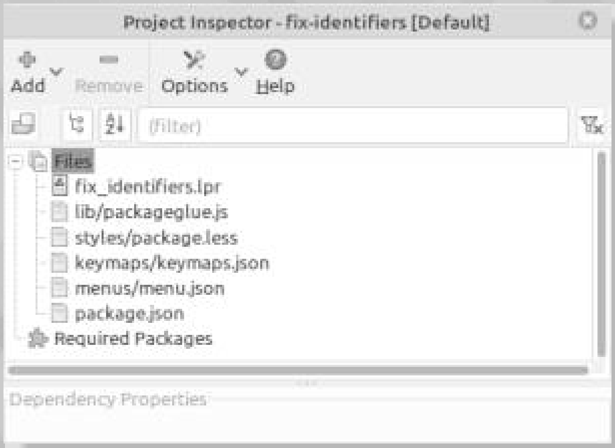
Activation Commands

Command Name	Scope
fix-identifiers	workspace

Figure 4: New project type: Atom Package

Once you confirm your choices, the IDE will create a project with several files (see figure 5 right top):

- **`fix_identifiers.lpr`**
  The program with generated code:
  it can be compiled.
- **`menu.json`**
  Atom menu entries: the menu entries here will appear in Atom under the 'Packages' menu.
- **`package.json`**
  The package manifest file.
- **`keymaps.json`**
  The keymaps offered by your package:
  It is necessary to edit the generated file, and assign a unique key combinations to each command.
- **`package.less`**
  CSS for your package if your code needs it.
  This will be loaded into the editor.
- **`packageglue.js`**
  This file contains the Atom package wrapper Javascript code shown above.
  You may edit this to your liking. If you change the name (or output file) of the Pascal project, you must manually change the name of the imported project file here.


Project Inspector - fix-identifiers [Default]

```
Files
  fix_identifiers.lpr
  lib/packageglue.js
  styles/package.less
  keymaps/keymaps.json
  menus/menu.json
  package.json
Required Packages
```

Figure 5: The new project

The most interesting is of course the program code. The class declaration is much as we expected:

```
TAtomFixIdentifiersApplication = Class(TAtomApplication)
  Protected
    procedure DoActivate(aState : TJSObject); override;
    procedure DoDeactivate; override;
    procedure DoSerialize(aState : TJSObject); override;
  Public
    Procedure FixIdentifiers;
end;
```

The interesting function is of course DoActivate, this is where we start the ball rolling. The new project wizard has already filled it with code:

```
procedure TAtomFixIdentifiersApplication.DoActivate(
  aState: TJSObject);
Var cmds : TJSObject;
begin
   inherited DoActivate(aState);
   cmds:=TJSObject.New;
   cmds['fix-identifiers:activate']:=@FixIdentifiers;
   subscriptions.add(Atom.Commands.Add('atom-workspace', cmds));
end;
```

The **`Atom.Commands`** object controls all the commands of the Atom editor. It has a method Add which needs a scope, and a Javascript object which has a set of properties: each property has the name of a command, and the property value is the function that must be called when the command is activated. In the code above, the command name is **`fix-identifiers:activate`** and the function is **`FixIdentifiers`**.

The result of the Add command is an Atom disposable: We add it to the Subscriptions so that when the package is unloaded, the disposable is freed.

An empty **`FixIdentifiers`** procedure has been generated by the wizard, we just need to fill it with code. To do what we want, we need to find a reference to the currently activeeditor, and the text buffer that it is actually editing. When we have the buffer, when can simply tell Atom to do a series of search and replaces for a set of words that we wish to correct the case for: The buffer API has a method for this.
The following is a possible implementation, with a limited list of keywords to replace:

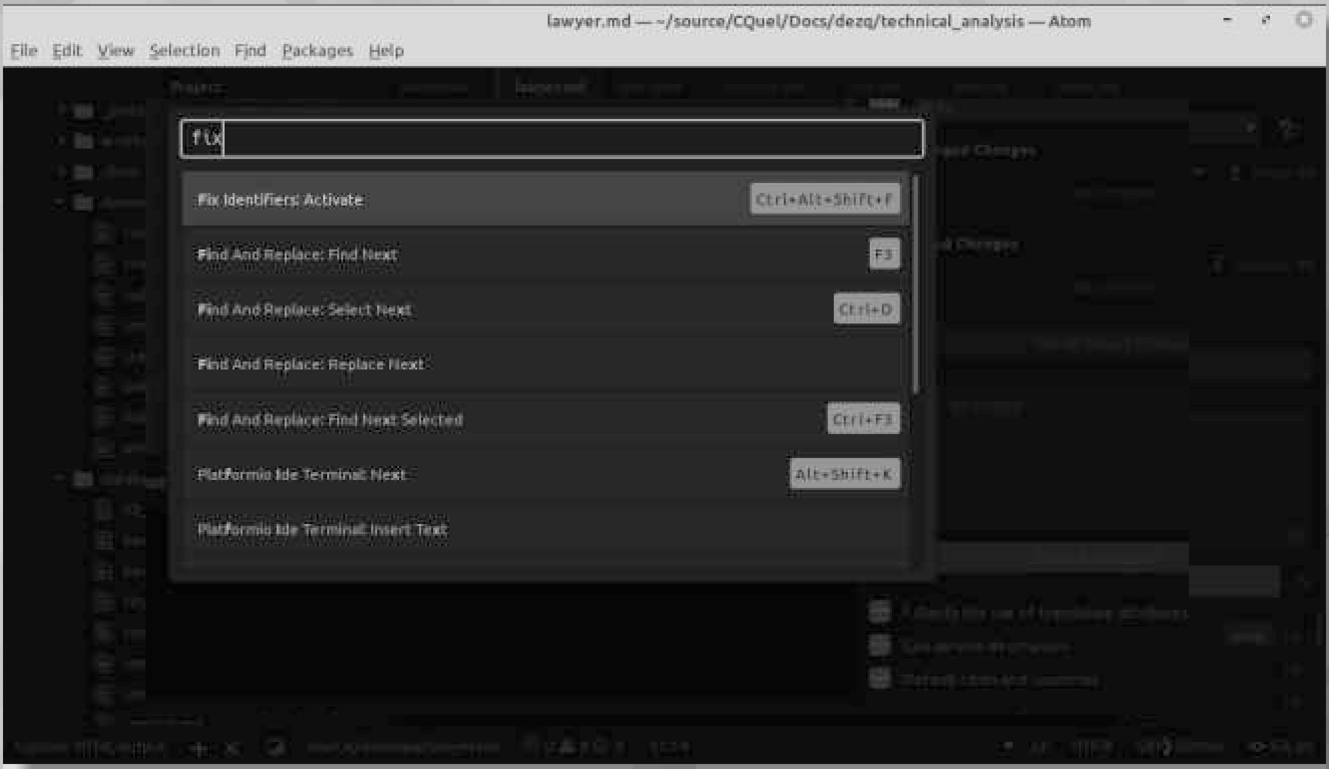
```
Procedure TAtomFixIdentifiersApplication.FixIdentifiers;
Const
   ToUpperCase : Array of string
                    = ('bigint','smallint','int','varchar',
                       'char','not null default','not null');
   SErrNoEditor =  'Cannot fix identifiers. No editor is active!';
   SErrNoBuffer =  'Cannot fix identifiers. No buffer available!';

Var
   Ed  : TAtomTextEditor;
   Buf : TAtomTextBuffer;
   S   : String;
   P   : Integer;
begin
   Ed:=Atom.WorkSpace.getActiveTextEditor;
   if not Assigned(Ed) then
   begin
     Atom.notifications.addWarning(SErrNoEditor);
     Exit;
   end;
   Buf:=Ed.getBuffer;
   if not Assigned(Buf) then
     begin
       Atom.notifications.addWarning(SErrNoBuffer);
       Exit;
     end;
   For S in ToUpperCase do
     DoUpperCase(Buf,S);
end;
```

We first get a reference to the active text editor: The WorkSpace object of the Atom editor manages the editors, and the **`getActiveTextEditor`** method of this object returns the currently active editor. This can of course be empty, and we display a nice notification if this is the case.
Once we have the editor, we get the underlying text buffer with **`getBuffer:`** because multiple editors can be editing the same buffer at the same time, the underlying buffer is a separate object of the editor. Normally the buffer cannot be empty, but for safety's sake we check for this and display a message if no buffer is found.

Once the buffer is found, we loop through our list of identifiers we wish to uppercase, and call **`DoUppercase`**, passing it the buffer and the keyword we wish to uppercase.
The **`TAtomTextBuffer`** object has search (and replace) methods: **`Scan, BackwardsScan`** and **`replace`** which allow us to do what we want.
Unfortunately, the replace method does not allow to use placeholders in the replacement text, so we opt for the **`BackwardsScan`** method. Using the backwards scan instead of the forward scan, this avoids the danger that the search algorithm gets stuck in an infinite loop, because the replacement text will also match the search expression.
The routine starts with creating a regular expression that will only match whole-word forms of the keyword, and a lowercase and uppercase version of the search term.

```
Procedure TAtomFixIdentifiersApplication.DoUppercase(
aBuf : TAtomTextBuffer;
aWord : String);
Var
   S,ARegex,aLower,aUpper : String;
   P : Integer;
begin
   aRegex:='(^|\W*)'+aWord+'(\W|$)';
   aLower:=LowerCase(aWord);
   aUpper:=UpperCase(aWord);
   aBuf.BackwardsScan(TJSRegexp.New(aRegex,'ig'),
   procedure(aMatch : TAtomBufferScanMatch)
   begin
      s:=aMatch.matchText;
      P:=Pos(aLower,LowerCase(S));
      S:=Copy(S,1,P-1)+aUpper
         +Copy(S,P+Length(aWord),Length(S)-P);
      aMatch.replace(S);
   end);
end;
```

Then it invokes the BackwardsScan option with a regular expression object (defined in the JS unit) and a callback: the callback is invoked for each matched item.
The callback receives an object that describes the match, and that contains a replace method to actually replace the found term in the text buffer: we must use it to replace the search term with the uppercase keyword, taking care that we match any non-letters before and after the keyword.
That's it.


lawyer.md — ~/source/CQuel/Docs/dezq/technical_analysis — Atom
File Edit View Selection Find Packages Help

fix

Fix Identifiers: Activate Ctrl+Alt+Shift+F
Find And Replace: Find Next F3
Find And Replace: Select Next Ctrl+D
Find And Replace: Replace Next
Find And Replace: Find Next Selected Ctrl+F3
Platformio Ide Terminal: Next Alt+Shift+K
Platformio Ide Terminal: Insert Text

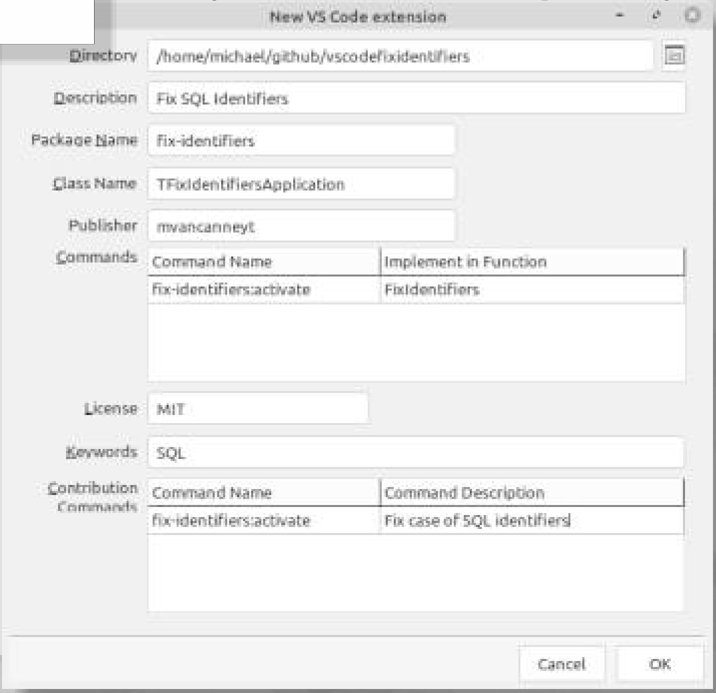
Figure 6: Our command in the command palette

Our plugin is ready. All that is left to do is assign a key combination to our command in the keymaps.json file:

```
{
  "atom-workspace": {
    "ctrl-alt-shift-f": "fix-identifiers:activate"
  }
}
```

We compile the lazarus project, and restart Atom. When we press **ctrl-shift-P** to invoke the command palette, we start typing the command name, we can see our command as in figure 6 on this page.
You can easily verify that the command actually changes the casing of the SQL keywords.
If you wish to debug the package, you must start the Atom editor with the '–dev' commandline option. When you do so, then you can show the Chromium 'Developer tools' using the 'View - Developer - Toggle Developer tools' menu: this will present you with the sources of your plugin - in Pascal - and you can debug the Atom package.
To distribute your package, all you need to do is create a .zip file from the directory with the code, or push it onto a github repository.

Figure 7: The VS Code Extension options dialog



Directory	/home/michael/github/vscodefixidentifiers
Description	Fix SQL Identifiers
Package Name	fix-identifiers
Class Name	TFixIdentifiersApplication
Publisher	mvancanneyt
License	MIT
Keywords	SQL

Commands

Command Name	Implement in Function
fix-identifiers:activate	FixIdentifiers

Contribution Commands

Command Name	Command Description
fix-identifiers:activate	Fix case of SQL identifiers

7 A SAMPLE VS CODE PACKAGE

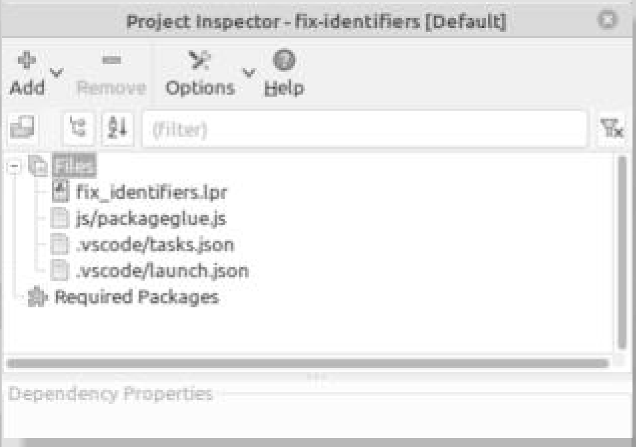
The functionality that was made for the Atom editor can of course also be implemented for VS Code. To do so, we can start the VS Code Extension in the Lazarus IDE's 'Project-New Project' dialog. figure 7 on page 24
The various items that can be entered for VS Code are - not surprisingly - very similar to the one in the Atom package dialog:

- **Directory**
  Every VS Code extension lives in its own directory. Here you specify the directory for the new extension.
- **Description**
  A textual description of your package, it goes in the manifest file (package.json).
- **Package Name**
  A (unique) name for your package, it goes in the manifest file.
- **Class Name**
  The Pascal class name for the application class.
- **Publisher**
  If you want to publish your package in the online VS Code extension repository, here you must enter your developer name.
- **Commands**
  The commands that your package will provide to the editor. For each command you must specify a unique name, and the name of a pascal function that will be called when the command is invoked. Again, an empty function will be generated for each function you specify here.
- **License**
  The license for your package, it goes in the manifest file.
- **Keywords**
  Some keywords (space separated) for your package, it goes in the manifest file.
- **Contribution Commands**
  The commands that will cause your package to be loaded by the editor. The scope is a valid Atom scope identifier such as atom-workspace.
  This list goes in the manifest file. VS Code editor will use this list to present your commands in the command palette.

For the sample code, we use the same names and settings as in the Atom package.

When you click OK, the IDE will make a set of files that make up the extension (see figure figure 8 on page 25):

- `fix_identifiers.lpr`
  The program with generated code: it can be compiled.
- `.vscode/tasks.json`
  This file is used by VS Code to build your package: It contains the Pas2JS command-line needed to build your package; it is possible to edit and compile your code in VS Code.
- `.vscode/launch.json`
  This file is used by VS Code to run and debug your package: It contains the necessary command-line options needed to start VS Code with your extension loaded.
- `package.json`
  The package manifest file.
- `js/packageglue.js`
  This file contains the VS Code extension Javascript wrapper code shown above. As in the case of the Atom package: If you decide to change the name (or output file) of the Pascal project, you must not forget to change the name of the imported project file here.



Again, the IDE has generated a project file that is ready to be compiled, you just need to create some code in the correct callbacks.

```
TFixIdentifiersApplication = Class(TVSCodeApplication)
  Protected
    procedure DoActivate; override;
    procedure DoDeactivate; override;
  Public
  function FixIdentifiers(args : TJSValueDynArray) : JSValue;
end;
```

Note that the generated **FixIdentifiers** function has some arguments and returns a value. The **DoActivate** function contains the code to register our command:

```
procedure TFixIdentifiersApplication.DoActivate;
Var
   disp : TVSDisposable;
begin
   inherited DoActivate;
   disp:=VSCode.commands.registerCommand('fix-identifiers:activate',
   @FixIdentifiers);
   TJSArray(ExtensionContext.subscriptions).push(disp);
end;
```

This code looks very similar to the DoActivate code of the Atom package. The result of the **registerCommand** command is a VS Code disposable class: The **ExtensionContext** was passed by VS Code to the VS Code extension. It contains a **Subscription** array which you can push elements on: We push the disposable result of the **registerCommand** to the **Subscriptions** array.
The **FixIdentifiers** method has been generated by the Lazarus IDE wizard, and we must fill it with code to implement our plugin.

The API of VS Code is quite big, it is documented here:
https://code.visualstudio.com/api/references/vscode-api

Unfortunately, the VS Code editor does not offer a search and Replace API such as it exists in Atom. We must implement the search and replace ourselves.

To change the contents of a document, you must first obtain a reference to the document. We do this in a similar manner as in the Atom plugin: the **window.activeTextEditor** returns an active text editor: an object of class **TVSTextEditor**.

The Document property of a **TVSTextEditor** class is an in instance of **TVSTextDocument** which contains the actual document that the user is editing. This class does not offer methods to directly manipulate the text: every edit must be done by a **TVSTextEditorEdit** instance: the **TVSTextEditor** class has an edit method which creates such an edit and calls an event handler with the created instance (called **editBuilder** in the code below).
To make things easier, we will retrieve the whole text of the document (there is a method called getText for this), replace all identifiers in this text, and then use the **TVSTextEditorEdit's replace** method to set the new text of the document. The replace method replaces a given range's text with a new supplied text.

The replacing of the text needs a Range (class **TVSRange**): this is a small object that contains two positions: the start and end position of a range of text. Since we will be replacing the whole text of the document, we create a range that starts at row 0 column 0, and which is 1 line too long: the **validateRange** method of the **TVSTextDocument** clips the range so it is valid, and we use that to correct the Range.
Putting all this together leads to the following code:

```
function TFixIdentifiersApplication.FixIdentifiers(
       args : TJSValueDynArray) : JSValue;
Const  SErrNoEditor = 'Cannot replace identifiers: no editor';
Var  Ed: TVSTextEditor; aText : String; R: TVSRange;
begin
  Result:=null;
  Ed:=VSCode.window.activeTextEditor;
  if not Assigned(Ed) then
  begin
    VSCode.window.showInformationMessage(SErrNoEditor);
    exit;
  end;
  aText:=Ed.document.getText();
  aText:=DoUppercaseSQL(DoUpperCaseSQL(aText));
  R:=TVSRange.New(0,0,Ed.document.lineCount,0);
  R:=Ed.document.validateRange(R);
  Ed.edit(procedure (editBuilder: TVSTextEditorEdit)
  begin
    editBuilder.replace(R,aText);
  end
  );
end;
```

The **DoUppercaseSQL** does the actual search and replace on the text. It is a simple loop which uses the standard Javascript **String.replace** to do the search and replace.

```pascal
Function TFixIdentifiersApplication.DoUpperCaseSQL(aText : String) : String;
Const
ToUpperCase : Array of string  = ('bigint','smallint','int','varchar',
                                  'char','not null default','not null');
Var  S,aRegex,aRepl : String;
begin
  Result:=aText;
  For S in ToUpperCase do
  begin
    aRegex:='(^|\W*)'+S+'(\W|$)';
    aRepl:='$1'+UpperCase(S)+'$2';
    Result:=TJSString(Result).replace(TJSRegexp.New(aRegex,'ig'),aRepl);
  end;
end;
```

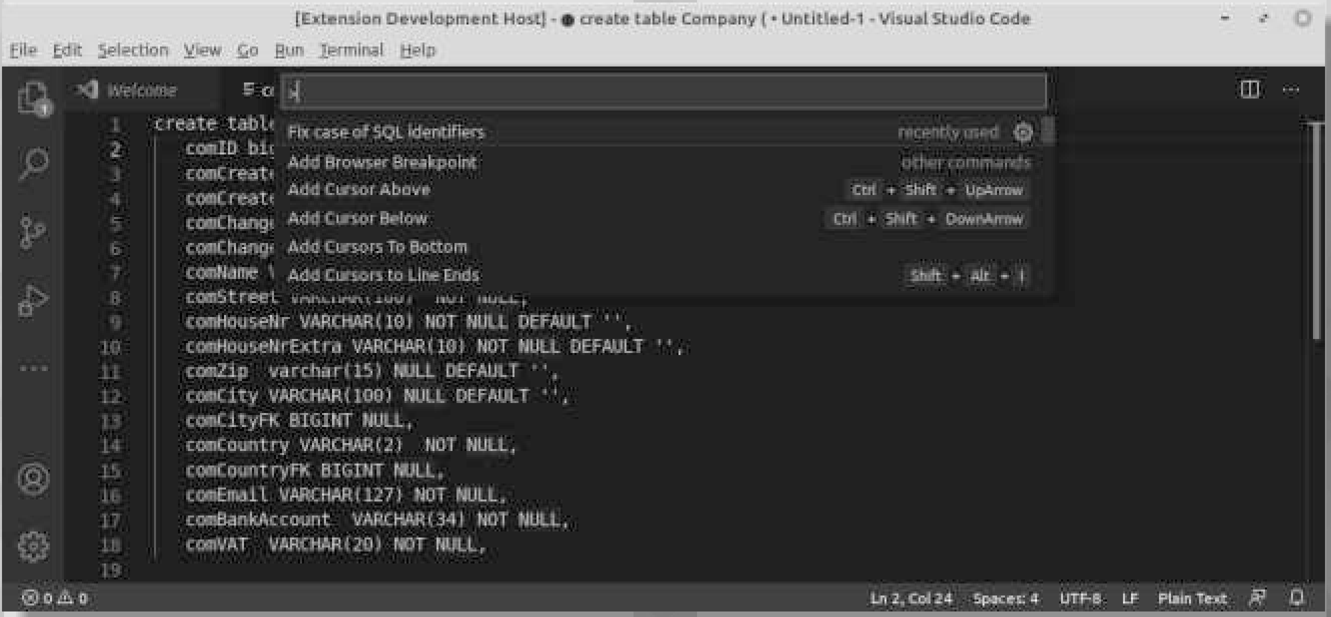
This is maybe not the most efficient algorithm, but it will do nicely for demonstration purposes. Compiling the program and debugging it in VS Code we can see that the project actually works:

If you want to distribute your package, you need to build it. In order to do so you need to install the vsce (Visual Studio Code Extensions) npm package:

```
npm install -g vsce
```

This will install a vsce command on your system, which you can then use to create a .vsix file:

```
vsce package
```



```
create table
    comID bi
    comCreate
    comCreate
    comChang
    comChang
    comName
    comStreet
    comHouseNr VARCHAR(10) NOT NULL DEFAULT '',
    comHouseNrExtra VARCHAR(10) NOT NULL DEFAULT '',
    comZip  varchar(15) NULL DEFAULT '',
    comCity VARCHAR(100) NULL DEFAULT '',
    comCityFK BIGINT NULL,
    comCountry VARCHAR(2)  NOT NULL,
    comCountryFK BIGINT NULL,
    comEmail VARCHAR(127) NOT NULL,
    comBankAccount  VARCHAR(34) NOT NULL,
    comVAT  VARCHAR(20) NOT NULL,
```

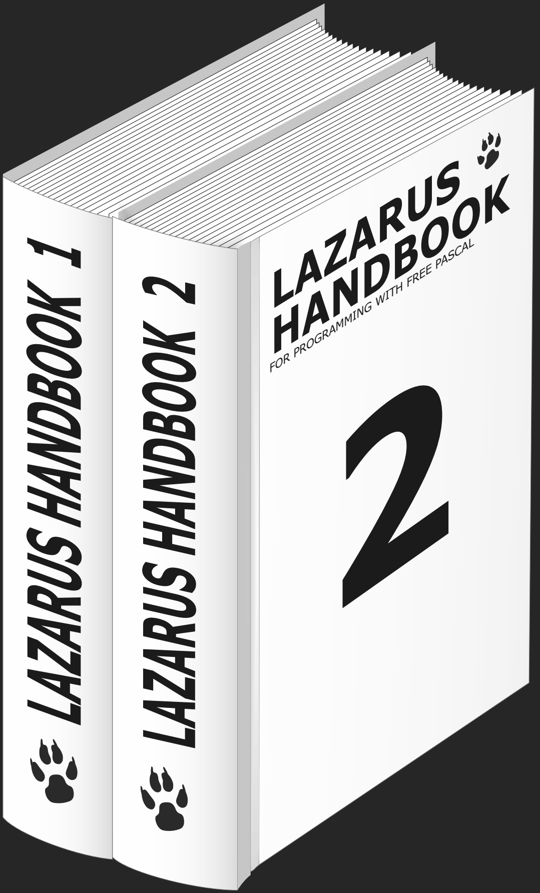
Figure 7: Our command in the command palette

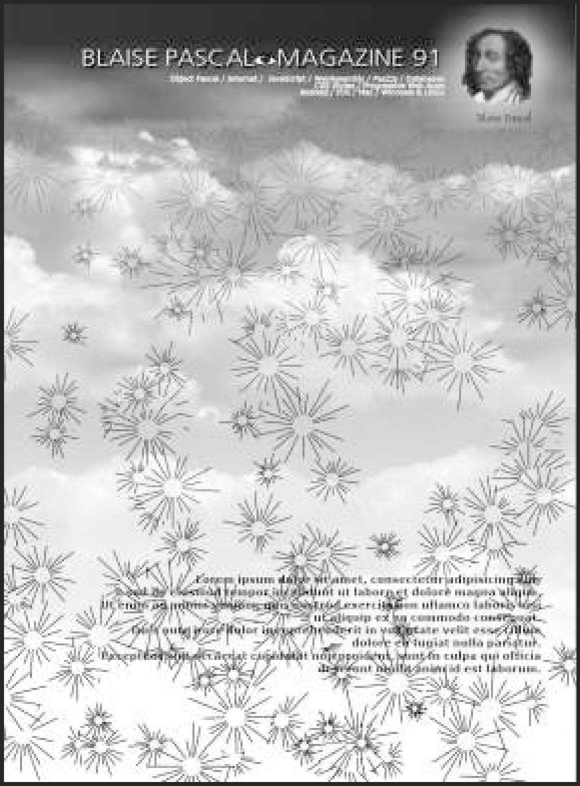
The packager may complain about a missing repository, but if all went well you will get a message that your file was created:

```
home:~/github/vscodefixidentifiers> vsce package
DONE Packaged:
/home/michael/vscodedemo/
fix-identifiers-0.0.1.vsix (17 files, 365.06KB)
```

8 CONCLUSION

VS Code and Atom plugins are normally created in Javascript**. Thanks to Pas2JS and TMS Web core**, Pascal programmers can now program plugins for these two popular engines in Pascal. The method shown here will become more simple in the future: when library support is finished, then the glue code will no longer be necessary.





Subscription Combi (4)

Subscription + Lazarus Handbook

Ex Vat 9%
Including shipment !

BITMAP ENLARGEMENT BY INTERPOLATION

By David Dirkse

starter expert

INTRODUCTION

The Delphi TBitmap class has a two dimensional array of bytes,words or cardinals called the canvas which holds an image.
An individual byte,word,cardinal of this canvas is called pixel.
On many occasions images need to be resized.
This may be done by copying the bitmap to a new one of different size,
using a suitable algorithm to calculate the pixel values of the new bitmap.
One method is to take the individual pixels from the destination bitmap,
project them over the source pixels and calculate the average color of the source pixels covered. I call this the "projection method".

This method works fine in case of image reduction.
For images enlarged by a factor 2 or 3, the result is not smooth. In this article I describe a better algorithm for these cases:

- transfer the source pixels directly to their new position in the destination bitmap
- use interpolation to calculate the remaining -in between- pixels of the destination bitmap.
  I call this the "interpolation method".

See the next pictures showing the different resizing algorithms for a threefold magnification:

Figure1: Original

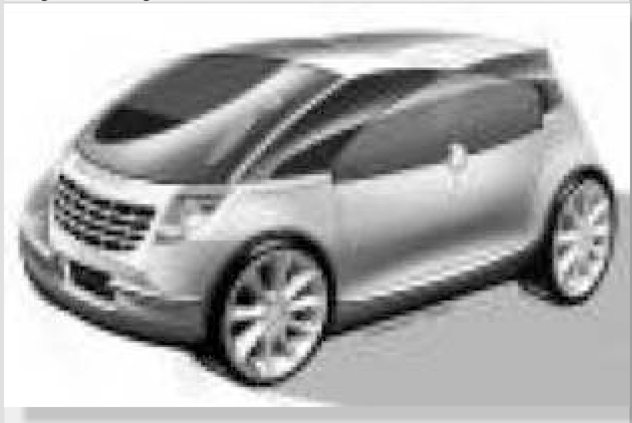


Figure 2: Projection method

Tested with: Delphi 7

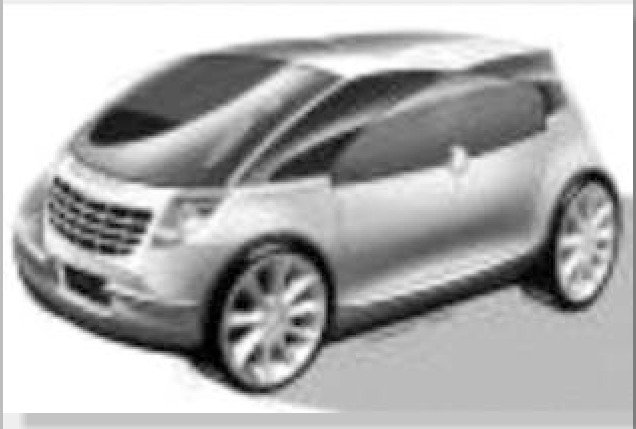
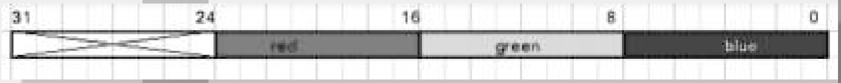


Figure 3: Interpolation method

Please use a magnifying glass to see the differences more clearly.

TBitmap class

In this project, pixels are cardinals only (unsigned 32 bit integers).
This is the internal layout of a 32 bit pixel:



There are 8 bits per color, color intensity ranges from `0` to `255`.
Bits `24..31` are not used.

Next picture shows the coordinate positions of the pixels.
The scanline[`y`] property supplies the pointer to the first pixel of row y.

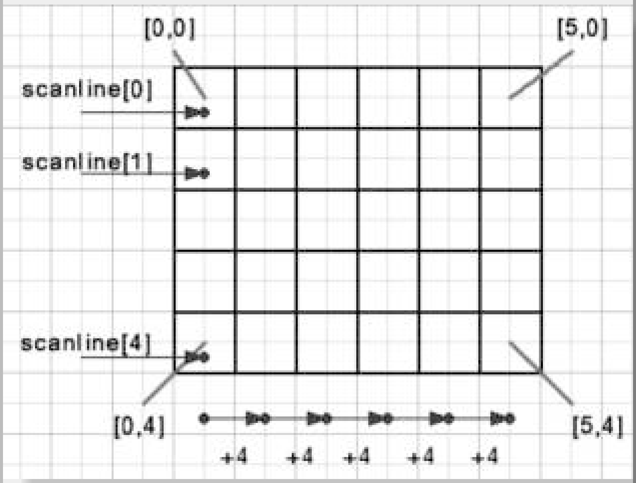


Figure 4: Copordinate positions

Pixels on the canvas of bitmap map are addressed by: `map.canvas.pixels[x,y]`
This is a slow process, only suitable for a few individual pixels.
Many times faster (50*) is to address pixels by a pointer to their memory location.
To facilitate pointer calculations I store these pointers as dwords.
Next a bitmap named map is created with 100 rows and 200 columns:

```
type PDW = ^dword;
...
var map : TBitmap;
  p0  : dword;       // pointer to [0,0]
  pstep : dword; // pointer distance between rows
....
begin
   map := TBitmap.create;
   with map do
   begin
      width := 200;
      height := 100;
      pixelformat := pf32bit;
   end;
   p0 := dword(map.scanline[0]);
   pstep := p0 - dword(map.scanline[1]);
```

Now the expression

```
   color1 := map.canvas.pixels[12,75];        //----1
```

can be replaced by

```
   color1 := PWD(p0 - 75*pstep + (12 shl 2))^; //----2
```

Note:
In cases `-1-` and `-2-`
before the values of `color1` are different.
In case `-1-` the `red` field occupies bits `0..7`, `blue 16..23`.
This is the **Windows color format** for `32 bit` and `24 bit` pixels.

Regarding a `100*200 pixel` bitmap as a one-dimensional `array[0..19999] of dword`, the first `dword [0]` is at pixel position `[0,199]`, the left bottom.

`Dword [1]` is at pixel position `[1,199]` which is `4` bytes higher.
Going from [0,199] to [0,198]
requires addition of `4*200 = 800` to a `pointer`.
`Pointers` are byte addresses.
`Expression (12 shl 2)` is a fast way to multiply 12 by 4.
For a next row, a pointer has to be subtracted by value `pstep = 4*column count`

Note:
Regarding a bitmap as a `one dimensional array A`,
the pointer to `A[0]` is `bitmap.scanline[bitmap.height-1]`.

Multiplication by 2
`A 5*5` bitmap is magnified to `10*10`.

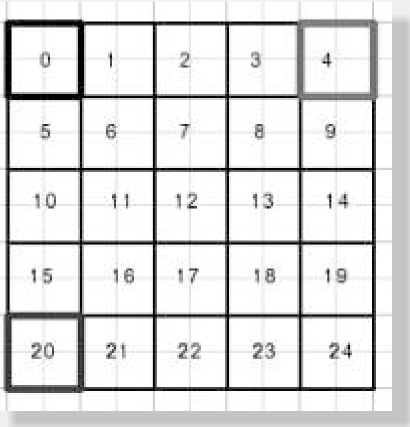


Figure 5: Coordinate positions

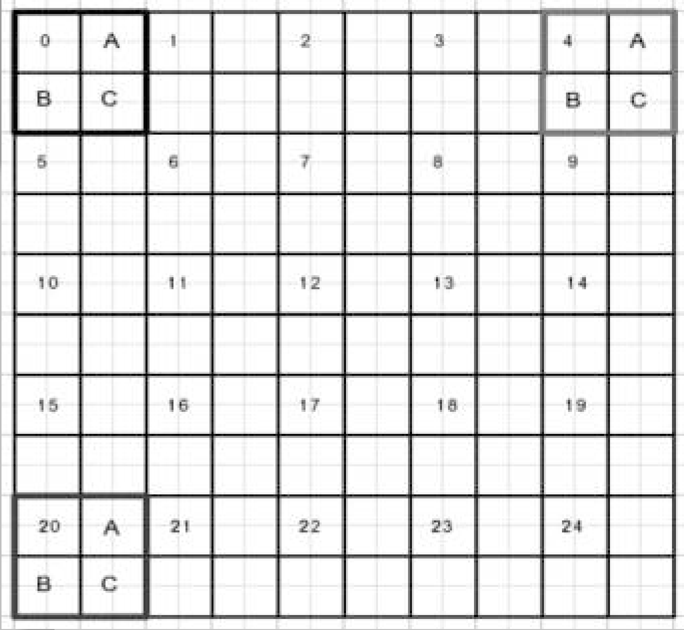


Figure 6: The pixels that are directly copied are indicated by a number.

Next the -in between- pixels A,B,C have to be calculated.
The bottom row and also the right column need separate action.

```
procedure X2copy
type TAIP = arry[1..8] of dword;
var c1,c2,c3,c4 : dword; //colors of source map
  AIP : TAIP;            //interpolation pixels
  pd0 : dword;           //destination row 0 pointer
  pd1,pd2 : dword;       //destination row pointers
  pdstep : dword;        //destination row difference
  ps0 : dword;           //source row 0 pointer
  psstep : dword;        //source row difference
  x,y : word;            //source pixel addressing
  py,py1 : dword;        //scratch pointers
```

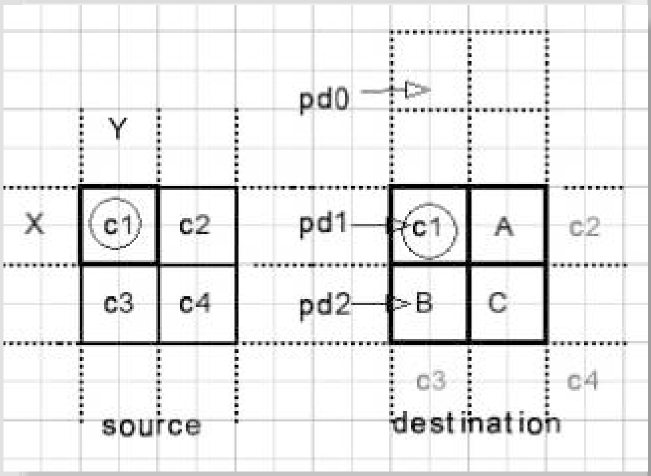


Figure 6: Pixels are processed starting left top to right bottom.

Variables **x,y** address the source pixel **c1** {c stands for color}.
C1 is copied directly to the destination bitmap.
To calculate the in-between pixels A,B,C the procedure interpolate24(AIP,c1,c2,c3,c4); is called.

A is AIP[1]
B is AIP[2]
C is AIP[3]

Interpolate24...{2x magnification, 4 variables} calls

```
procedure unpackColor(var r,g,b : byte; col : dword);
begin
  b   := col and $ff;
  col := col shr 8;
  g   := col and $ff;
  col := col shr 8;
  r    := col and $ff;
end;
```

which extracts the 8 bit r,g,b values from **dword col.**
c1 has **r1,g1,b1** values for red, green, blue.
c2 has **r2,g2,b2** values...etc.

Then r,g,b values are calculated for each A,B,C color.

```
A = (C1+C2)/2              for  r,g,b
B = (C1+C3)/2              for  r,g,b
C + (C1+C2+C3+C4+3)/4      for  r,g,b
```

Finally **r,g,b** colors are packed in **AIP[1], AIP[2}**..etc by a call to

```
procedure PackColor(var col : dword; r,g,b : byte);
begin
  col := ((r shl 16) or (g shl 8) or b);
end;
```

Please refer to the source code for details.

RIGHT COLUMN

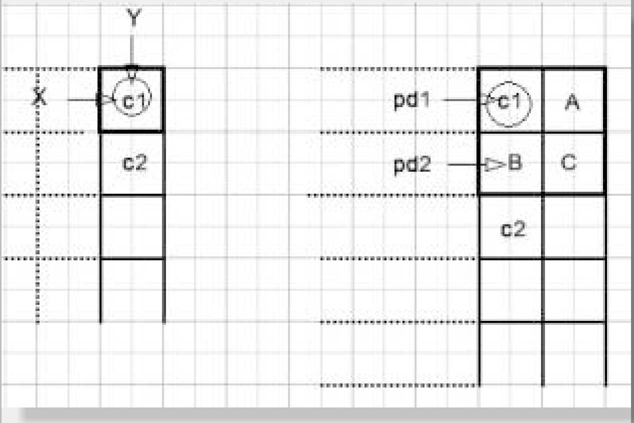


Figure 7: Coordinate positions

Pixel **c1** at **[x,y]** is copied to the destination bitmap.
Pixel **A** is equal to **c1**.
Pixels **B** and **C** are the average of **c1** and **c2** which is calculated by

```
procedure interpolate22(var AIP : TAIP; c1,c2 : dword);
//return AIP[1]
var r,g,b,r1,g1,b1,r2,g2,b2 : byte;
begin
  UnpackColor(r1,g1,b1,c1);
  UnpackColor(r2,g2,b2,c2);
  r := (r1 + r2) shr 1;
  g := (g1 + g2) shr 1;
  b := (b1 + b2) shr 1;
  packcolor(AIP[1],r,g,b);
end;
```

BOTTOM ROW

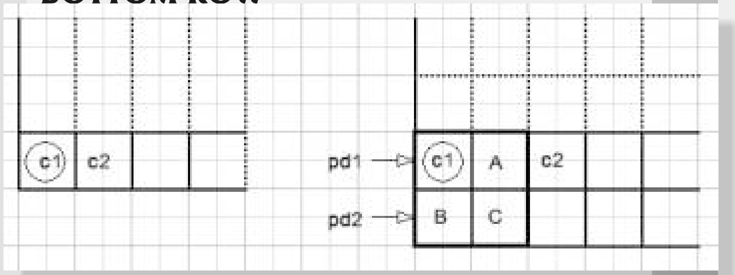


Figure 8: Bottom row positions

c1 is copied from the source bitmap.
B equals c1.
A and C are the avarage of c1,c2 calculated similar to the right column pixels.

Multiplication by 3
Below is pictured a 3x4 bitmap and its magnification by 3.

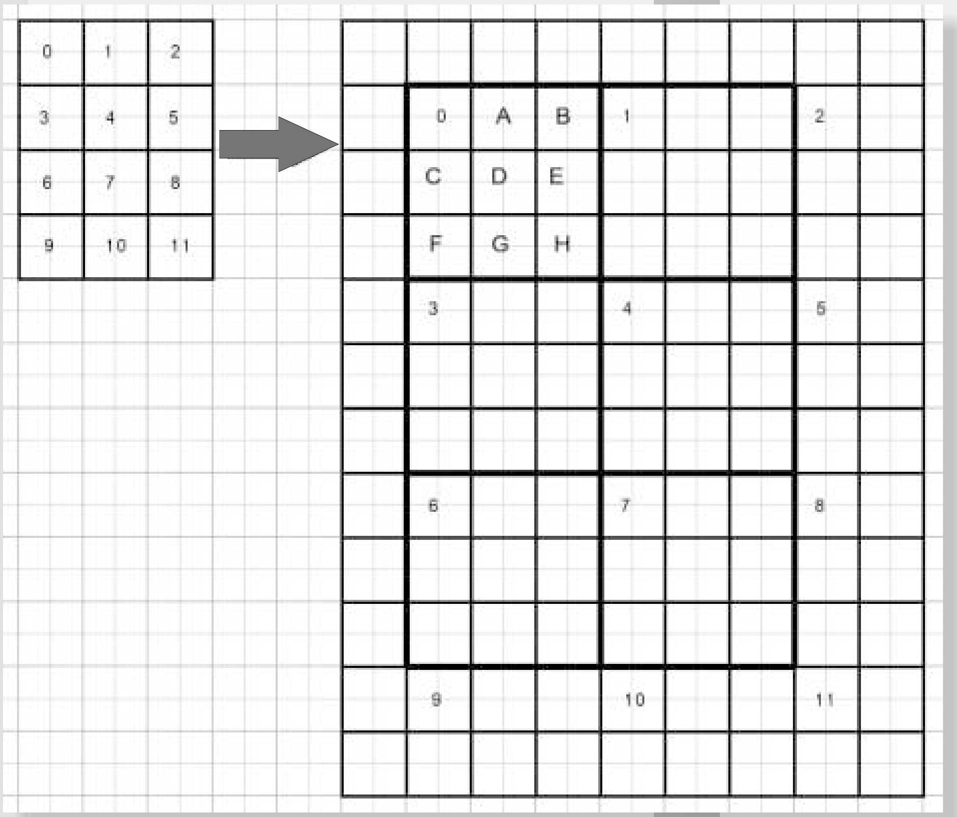
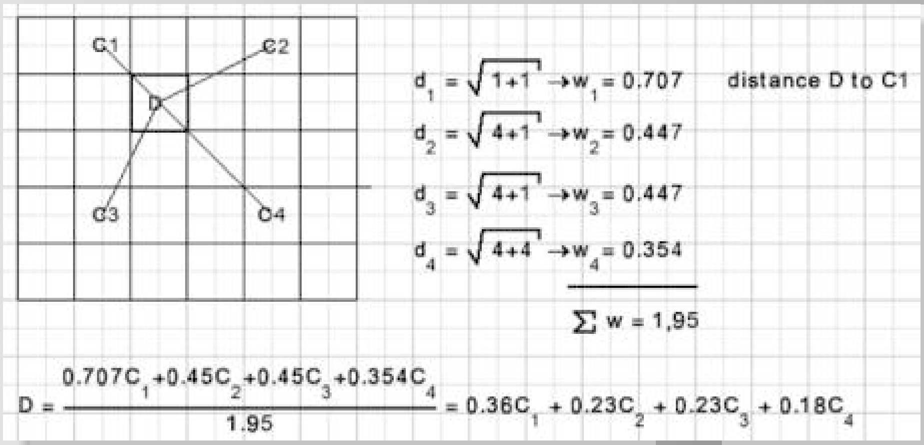


Figure 9: Magnification by 3

The process is similar to the x2 magnification however more calculation is required.
First the 3x3 pixelfields in the destination bitmap are processed.
Interpolation is more complicated.
I calculate pixels A,B,C,D,E,F,G,H as weighted average of C1,C2,C3,C4.
If a color (C1,C2..) has distance d to a pixel (A,B,C...) its weight factor w = 1/d.
The Pythagoras lemma is used to calculate the distances.

```
A = (1.C1 + 0.5C2)/(1+0.5) = 0.66C1 + 0.33C2
B = (0.5C2 + 1.C1)/(1+0.5) = 0.33C1 + 0.66C2
D = 0.36C1 + 0.23C2 + 0.23C3 + 0.18C4
```

Please look at the next picture: *(Figure 10)*



$$d_1 = \sqrt{1+1} \rightarrow w_1 = 0.707$$
$$d_2 = \sqrt{4+1} \rightarrow w_2 = 0.447$$
$$d_3 = \sqrt{4+1} \rightarrow w_3 = 0.447$$
$$d_4 = \sqrt{4+4} \rightarrow w_4 = 0.354$$
$$\sum w = 1{,}95$$
$$D = \frac{0.707C_1 + 0.45C_2 + 0.45C_3 + 0.354C_4}{1.95} = 0.36C_1 + 0.23C_2 + 0.23C_3 + 0.18C_4$$

Figure 10: Shows the Formula

Of course the calculation for D is repeated 3 times: for red, green and blue.

RIGHT COLUMN

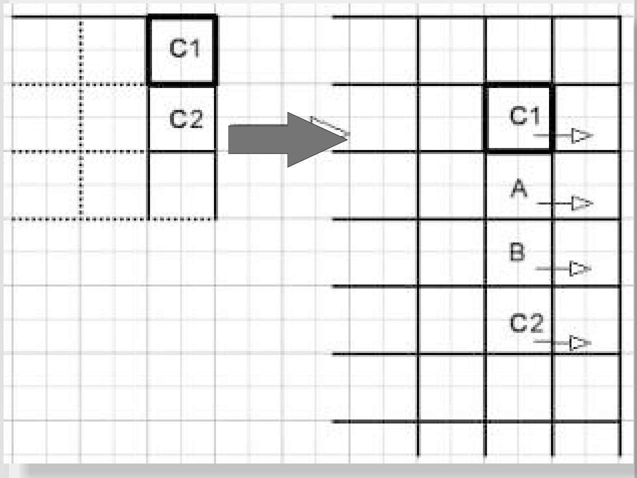


Figure 11: Right column
C1, C2 and interpolation colors A,B are copied also to the far right destination column

BOTTOM ROW

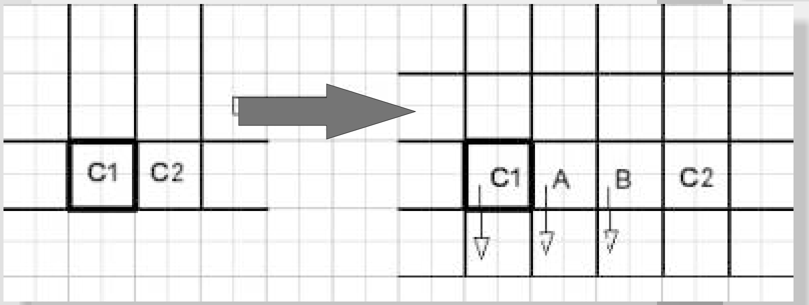


Figure 12: Bottom Row

The C1 and C2 values are copied directly.
A, B are interpolation colors. C1, C2, A, B are copied to the bottom row of the destination map.

SHOWING RESULTS

`map1` (loaded from disk), is displayed in paintbox1. Paintbox1 has a fixed size of `400*400` pixels.
`map2` is the result of expansion and this bitmap is displayed in paintbox2. This paintbox is `800*800` pixels in size. To show all pixels in case `map2` is larger, horizontal and vertical scrollbars are added on `form1`.

The scrollbar max property has to be adjusted for the size of map2.
The following code takes care

```
var d : smallInt;
...
begin
   d := map2.width - paintbox2.Width;
   if d < 0 then d := 0;
   Hscrollbar.max := d;
   Hscrollbar.position := 0;
   d := map2.Height - paintbox2.Height;
   if d < 0 then d := 0;
    Vscrollbar.Max := d;
   Vscrollbar.position := 0;
end;
```

A scrollbar **onChange** event calculates the rectangle to be copied from **map2** to **paintbox2**:

```
procedure TForm1.VscrollbarChange(Sender: TObject);
//V,H scrollbar changes
//repaint paintbox2
var BW,BH : word;//paintbox width,height
   rs,rd : Trect; //source,destination rect
begin
   BW := paintbox2.Width;
   BH := paintbox2.Height;
 with rs do
 begin
   left := Hscrollbar.position;
   top := Vscrollbar.position;
   right := left + BW;
   bottom := top + BH;
 end;
 with rd do
   begin
   left := 0;
   top := 0;
   right := BW;
   bottom := BH;
   end;
   paintbox2.Canvas.CopyRect(rd,map2.Canvas,rs);
end;
```

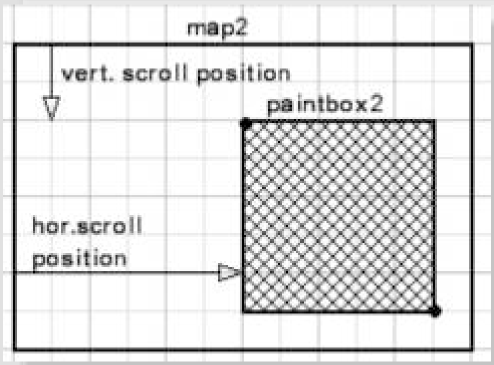


Figure 13: Bottom Row

To conclude I show another example of a 3 times magnified image using both the projection and the interpolation method:

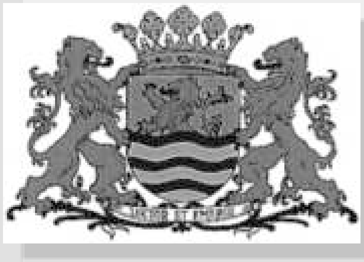


Figure 14: Original

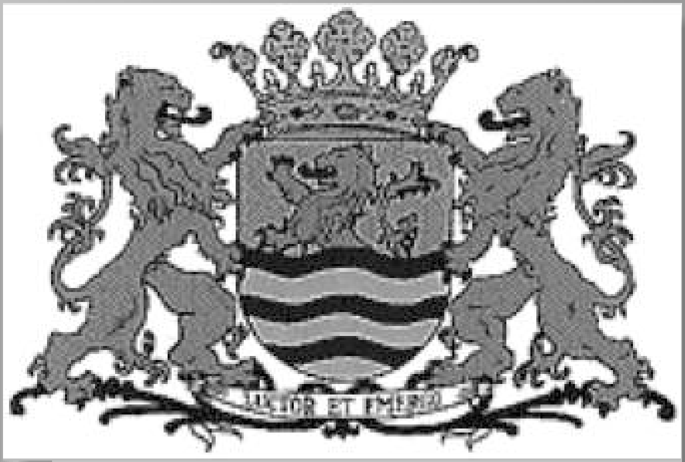


Figure 15: Projection method

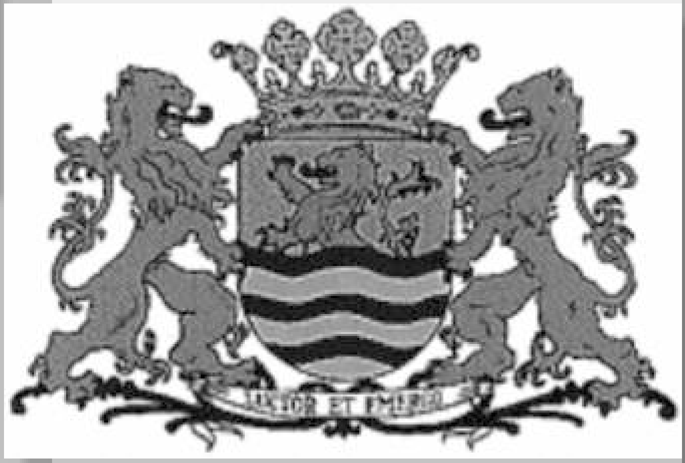
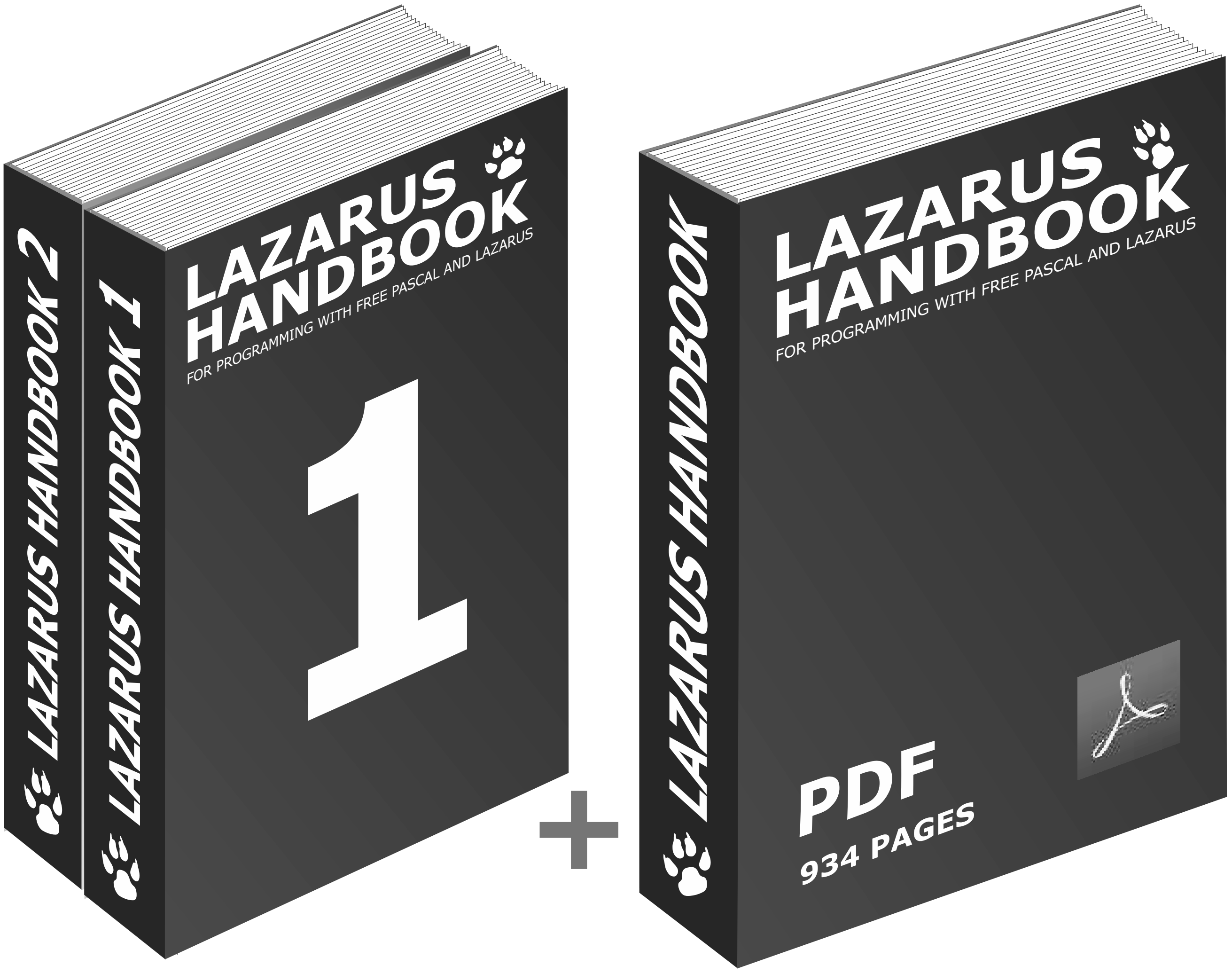


Figure 16: Interpolation method

The image represents the escutcheon of the Dutch county of Zeeland.
A translation of the Latin text **"Luctor Et Emergo"** is :
"*I struggle but I'll survive*".



Sewn POCKET (2)
60 euro
ex Vat including shipment inc. PDF

LAZARUS HANDBOOK POCKET edition is also sewn, (see next page) to make sure you will not lose pages after a while. It is printed on 100 percent guaranteed FSC certified Paper

INCLUDED:
bookmark - creditcard - usb stick which contains the personalized pdf file of the book and the extra program files. So you have your electronic as wel the printed book in one product.

934 pages
in two books

For ordering go to:
`https://www.blaisepascalmagazine.eu/product-category/books/`

Cyclic Redundancy Checking /

By David Dirkse

starter expert

Because of some glitches in the same article of issue 90 we have republished it in this issue (91) extra.

INTRODUCTION

Cyclic Redundancy Checking (CRC) is a way to insure the integrity of data. During transmission data may be disturbed by atmospheric interference, bad contacts or other hardware failures. Also damaged magnetic media cause corrupted data. Data also may be manipulated by interested parties. The basic idea of CRC checking is to attach a unique number to the data.
This number is generated in the following way: Regard the data (message M) as one big binary number.

Choose a number k (called the key) and divide `M` by `k`, the remainder is `r`.
The quotient `Q` is discarded. The remainder `r` is called the checksum and this number is attached to the message,
`M = Q.k + r .`
`(Message = quotient  times key + remainder)`
Look at the next picture for the case of data transmission:

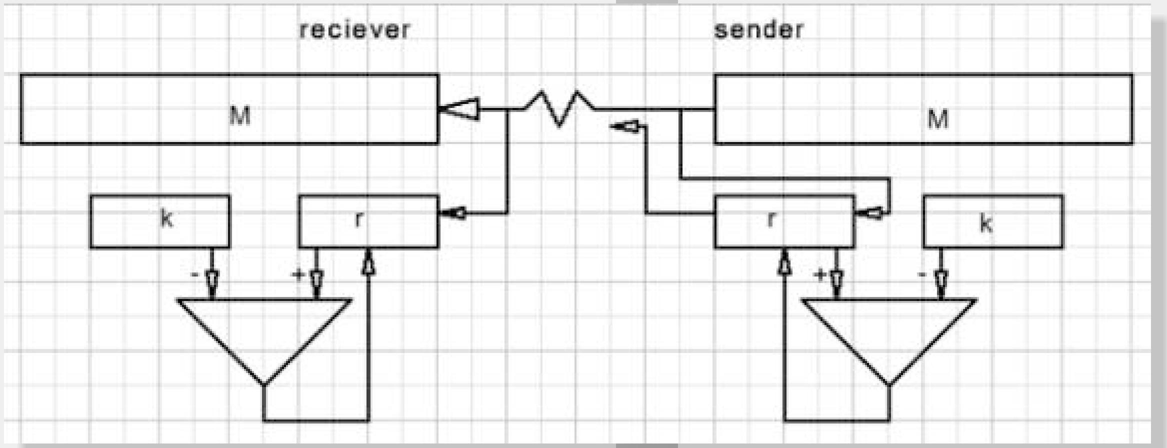


Figure 1: the case of data transmission:

At the sender side, bits of message M send are also shifted into register `r` while `k` is subtracted from `r`. After message `M`, the checksum `r` is transmitted.
At the receiver side also the checksum is generated.
When `M` bits are transmitted error free, the generated checksum at the receiver side equals the checksum send by the sender.

Tested with: Delphi 7

CAPABILITIES

In case of a message length of `32 bits` and a checksum of `16 bits`, there are `2^32/2^16 = 65536` messages that share the same checksum.
This looks bad at first glance but consider a randomly damaged message `M`: it only goes undetected if `1` out of `65536` checksums is generated.
With a n bit checksum the chances for a random error staying undetected is `1/2^n` .

DECIMAL EXPLANATION

To understand CRC checking, take the example of normal decimal arithmetic and a `key (k)` of `10`.
Then message `192743` generates a checksum of `3` regardless of the other digits.
Any error except for the last digit `3` goes undetected.
Reason is that `40, 700, 2000, 90000, 100000` are multiples of the key k.

Customized text

Now consider a key of
`11. 19274310  = 1218a111    {a = 10}`
This message will generate a checksum of `1`.
Next an **error** is imposed on `M`, which becomes `19574310 = 12407911` and now checksum `9` is generated. The error is detected.
Let our `key k` be `125`.

`8 * 125 = 1000` so, errors in digits `3,4,`...etc will go undetected because they impose errors on `M` that are a multiple of `1000`.
Here we reach an important conclusion:
an error `E` superimposed on message `M` goes undetected if `E` is a multiple of `k`.
Working decimal, `10n` may not be a multiple of `k for n = 1,2,3,….`
This is the case when the number system base and the key have no common factors.

With a key of k bits, producing checksums of `n = k-1` bits long, any single error burst within n bits will be detected.

PUBLIC OR SECRET KEY?
In the case of data transfers a public key, known by everyone, is fine.
Another situation occurs when **CRC checking** is used as a signature to protect data against unauthorized modification.
In this case the checksum is generated with a secret key, only known by the application.

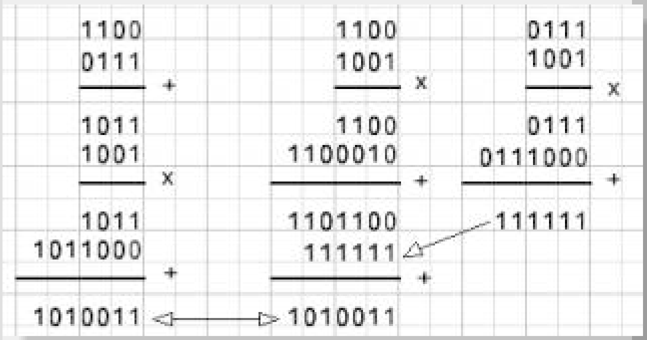


Figure 2: CRC cheking

SIMPLIFIED ARITHMETIC
Before, I mentioned that the checksum is generated by division.
Division implies borrows.
Borrows (and carries) may be avoided by defining addition (subtraction) as exclusive or (xor) operations. This simplifies hardware without reducing effectiveness.

Binary xor operations are
`0+0 = 0;  0+1=1;  1+0=1;  1+1=0;`
No carries, no borrows, they are simply ignored. There is no difference between addition and subtraction. xor is also called: “logical difference”.

In `x or , 0` is the unity element of operation:
`1+0 = 1,  0+1 = 1.`
1 is its own inverse : `1 + 1 = 0.`
The associative law holds:
`(a+b) + c = a + (b+c)`
The commutative law holds : `a+b = b+a`
The distributive law holds: `a(b+c) = ab + ac`
see the next example as proof where `a=1001`, `b=1100` and `c=0111`

The left column shows `a(b+c)`, the right column `ab + ac`.
So we have a valid arithmetic system.
Here is another reassuring example:
the calculation `a * b / b = a for a=010111` and `b=101101`

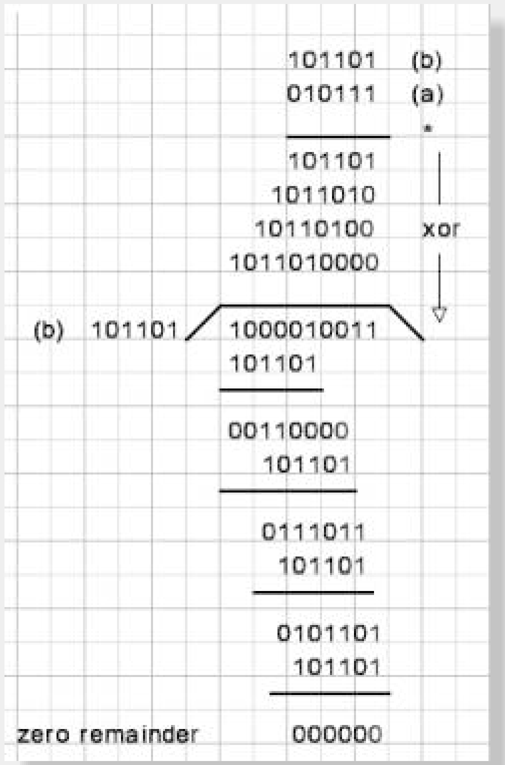
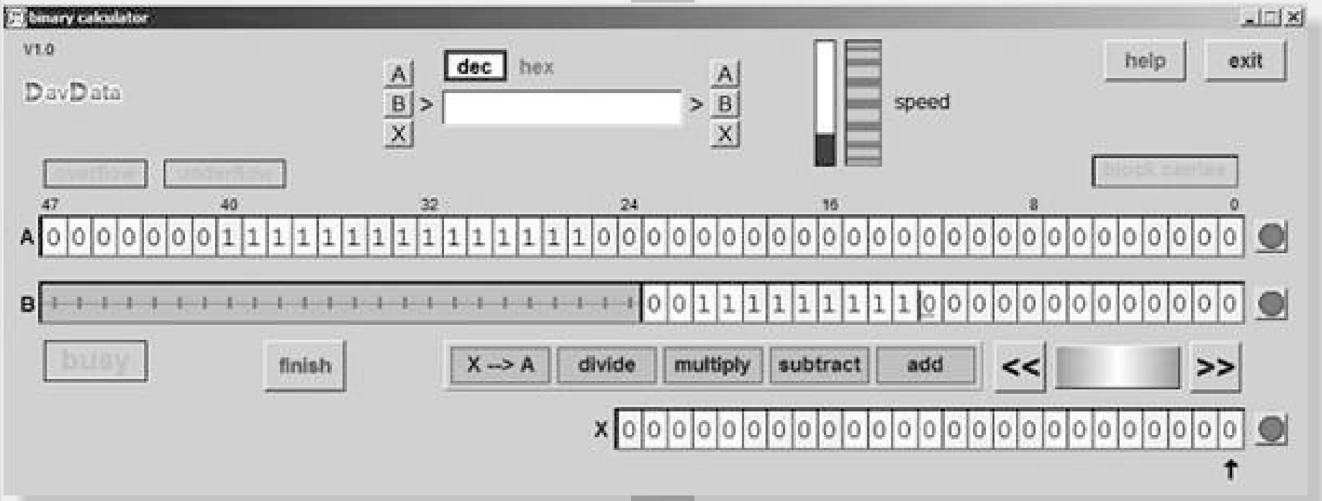


Figure 3: a valid arithmetic system.



```
A 000000011111111111111100000000000000000000000000
B 001111111110000000000000
X 0000000000000000000000000
```

Figure 4: a simple calculator

Note: we are not interested in the quotient.

CHOOSING A KEY

A key of `1 0000 00002` will generate checksums that equal the 8 least significant bits of `M` because any message of the form `xxx 0000 0000` is a multiple of the key.
Any other key is good.
The choice of a key depends on the type of errors expected. Many keys are designed to detect double bit errors that are far apart.
In that case errors like `100000000000000001` may not be a multiple of the key.
To choose a key may start with factorizing expected errors to make sure they are not a multiple of the key. This arithmetic, of course, must be performed using the above rules ignoring carries and borrows.

POPULAR KEYS

A popular `16 bit` key is
`1 0001 0000 0010 0001` which is known as the `X25` standard.
Another 16 bit key is
`1 1000 0000 0000 0101` called the CRC-16 protocol, used in **modems.**
The **Ethernet** standard uses the 32 bit key:
`1 0000 0100 1100 0001 0001 1101 1011 0111.`

The keys are designed to detect double bit errors that are many bits apart.
Choosing a certain key is based on the assumption that some errors are more likely to occur than others.

A BIT CALCULATOR

To explore and learn binary arithmetic with suppressed carries and borrows I have programmed a simple calculator. Below is a reduced image.

Calculations may be performed with or without carries.

The speed is adjustable from 1 to 25 operations per second.
It uses 3 registers:
`A : 48 bits.`
`B : 24 bits`, may be shifted left to represent value `B*2n` where n is the shift count.
`X : 25 bits.`

Operations are:
`Add:      A = A + B.` Customized text
Overflow sets if the sum exceeds `2^48 -1`
`Subtract: A = A - B`
Underflow is set when A becomes negative.
`Multiply: A = A + B.X`
(clear A before operation)
`Divide:   X =  A / B`
A holds remainder after division.

Numbers `(0,1)` may be entered directly into the register or entered in decimal or hexadecimal format in an edit box.
Register values may be displayed in decimal or hexadecimal format in this edit box.

Image below lists the controls and registers:

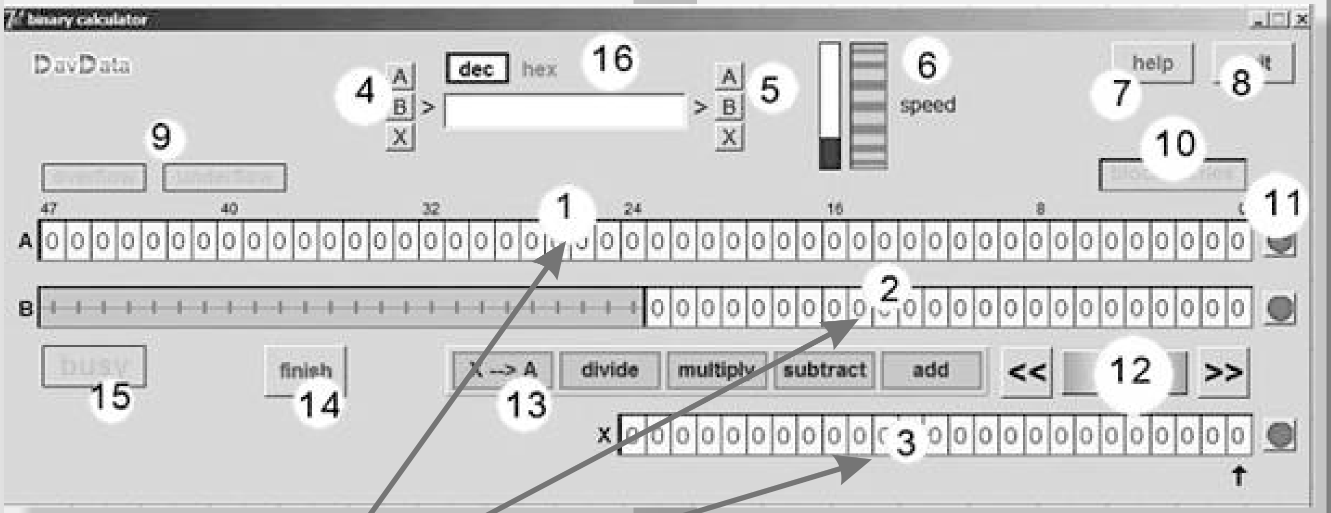


Figure 5: The controls and registers

1. **A register.**
2. **B register.**
3. **C register.**
4. **Buttons to transfer register to edit box.**
5. **Buttons to transfer edit box to registers.**
6. **Speed control**
7. **On-Line help**
8. **Close exerciser and exit.**
9. **Overflow / underflow indicators.**
10. **Carry suppression button.**
11. **Clear buttons for A,B,X registers.**
12. **B register left – right shift handles.**
13. **Operation buttons**
14. **Finish operation in progress, bypassing delays.**
15. **Operation busy indicator.**
16. **Edit component for decimal or hexadecimal display of numbers.**

THE DELPHI (7) PROJECT

`Unit 1` handles button events and indicators.
`Unit 2` consists of a class called `TEdit64` which is a descendant of `TObject`.
`TEdit64` has a box property pointing to a paintbox for display of register values.

The `TEdit64` class does not perform arithmetic operations, but only displays and edits data, shifts the B register and manipulates a cursor.
Data is held in property digits, an `array[…]` of byte.

Byte layout is:
`1xxx xxxx` right pointing arrow displayed at top of bit to indicate a borrow
`x1xx xxxx` left pointing arrow displayed at top of bit to indicate a carry
`xx10 xxxx` bit displayed in red color (not black). Not used.
`xxx1 xxxx` bit displayed bold. (not used)
`xxxx nnnn` number `0..15` .
Only `0,1` used here.
Some properties of TEdit64:

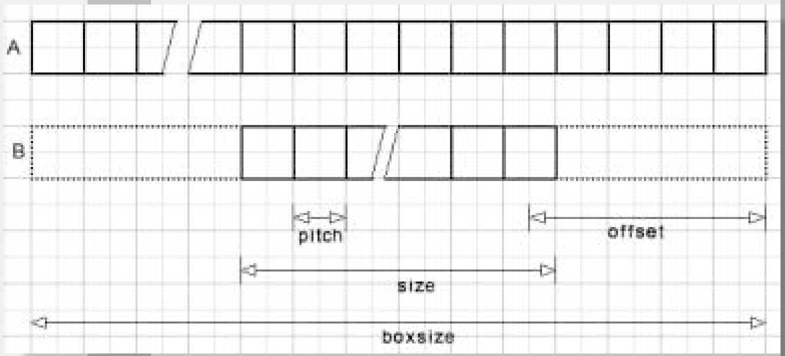


Figure 6: Some properties of TEdit64:

Size, boxsize and offset are counted in binary digits.
Pitch is the pixelcount per digit.
Binvalue: read (write) register value as 64 bit integer.
Box: associated paintbox for display of register.

`Unit 3` handles arithmetic operations. Operations are performed bit-wise.
A `rotationbutton` component (simulation of a rotary button as found in laboratory equipment) allows for adjustable speed.

The button is set for values `0..20`.
The delay is obtained from a microseconds timer component.
Value `0` causes `1Hz` operation speed, `20` causes `25Hz`.
Each step must cause the same relative speed increment so `25 = step20`.
`Step = e0.05.ln(25) = e0.1609`.
So, `delaytime = 1e6 / exp(n*0.1609)` where `n` is the `rotation-button` position.
This unit has the following procedures:
`procedure ADDstep(n : byte);`
Adds bit `n` of `B` to corresponding bit of `A`.
`procedure carrystep(n : byte);`
is called to propagate a carry through `A`.
`procedure AddBtoA;`
calls `ADDstep` and `carrystep` to make full addition `A = A + B`.

A similar case is subtraction with `procedures subtractstep(n:byte)`, `borrowstep(n:byte)` being called by `procedure subtractBfrom A`.
Multiplication is done by calling `AddBtoA` repeatedly if the corresponding `X` bit is `1`.
`Division` calls `subtractBfrom A` repeatedly, setting an `X` bit if subtraction is done.
For each position `(offset)` of B a check is needed because subtraction may not be possible.
Different checks are used for the case of carries and carry suppression.

`function AminBOK: boolean;` returns true if subtraction with carries is possible.
Note: in computer hardware the subtraction is actually made and negative results are restored. However, non restore divide algorithms also exist.
`function AxorBOK: byte;` checks for carry suppressed division and returns
- 0 : continue shifting B register right
- 1 : xor B to A and shift B right
- 2 : error

See examples below:

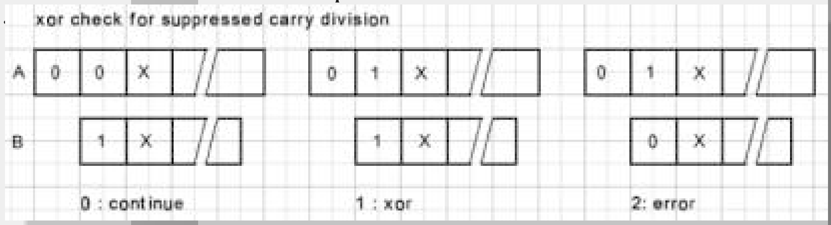
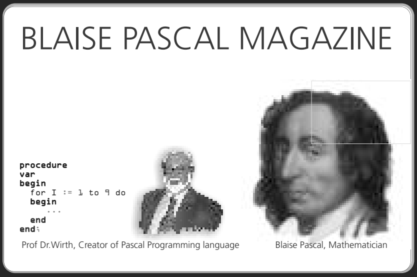
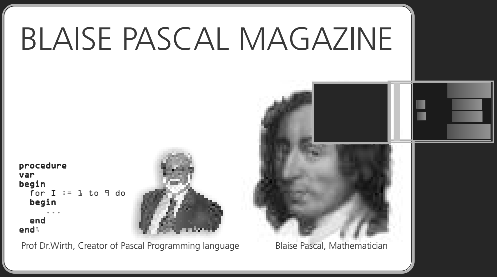


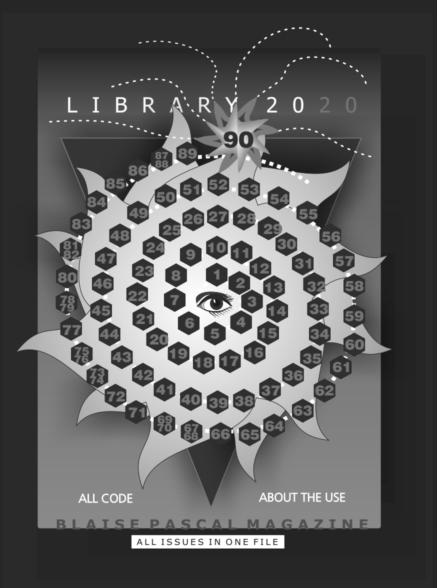
Figure 7: Examples
Please refer to the source code for details.
The **`rotation button`** *component was described some years ago in Blaise Pascal Magazine.*
For the **`microseconds timer and the array button component`** *(used to select operations) see my book:* "Computer Math and Games in Pascal".
The buttons and extra examples are available via download from your registered subscription address:

`https://www.blaisepascalmagazine.eu/your-downloads/`


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

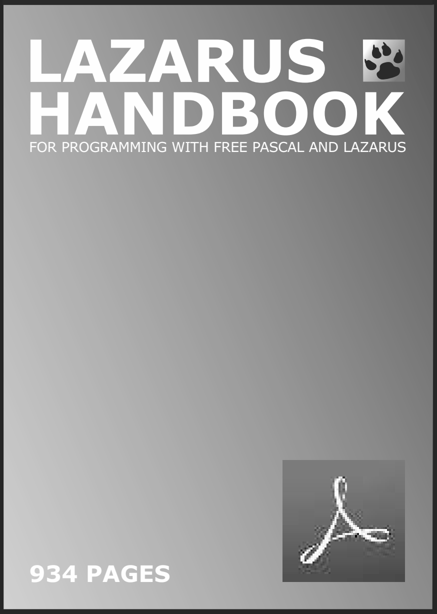
Editor in Chief: Detlef Overbeek
Edelstenenbaan 21 3402 XA
IJsselstein Netherlands
Prof Dr.Wirth, Creator of Pascal Programming language
editor@blaisepascalmagazine.eu
https://www.blaisepascalmagazine.eu

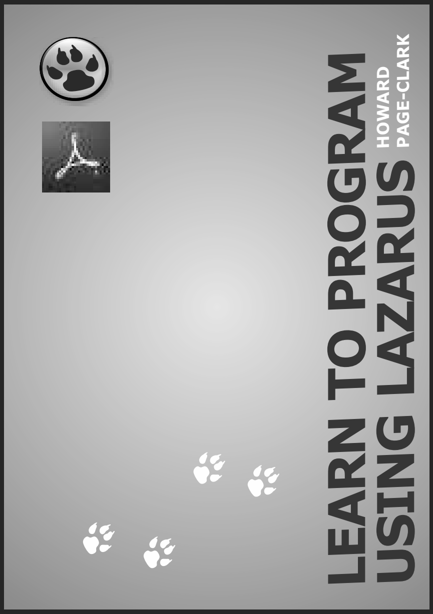

BLAISE PASCAL MAGAZINE
procedure
var
begin
for I := 1 to 9 do
begin
...
end
end;
Prof Dr.Wirth, Creator of Pascal Programming language
Blaise Pascal, Mathematician



SUPER OFFER (5)
€ 150 ex Vat
free shipping

BLAISE PASCAL MAGAZINE 89
Object Pascal / Internet / JavaScript / WebAssembly / Pas2Js / Databases
CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux
LAZARUS 2.010 FOR WEB:
WEbCore for FREE, for Linux for Mac, and Windows,
a special to install a free groep of components
DELPHI - DELEAKER
helps to find memory leaks in Delphi
MYCLOUD DATA
Delphi server client app: an authors and booklist
available via the internet
CODE SNIPPETS for Delphi and or Lazarus creating buttons
and fields that can switch color
depending on handling content
UNTYPED I/O
These records may be
transferred without bothering about
their internal structure.





1. One year **Subscription**
2. **The newest LIB Stick**
   - including Credit Card USB stick
3. **Lazarus Handbook** - Personalized
   -PDF including Code
4. Book **Learn To Program** using Lazarus PDF
   including 19 lessons and projects
5. **Book Computer Graphics Math & Games**
   book + PDF including ±50 projects

THE CRISPR PAGES PAGE 1/2

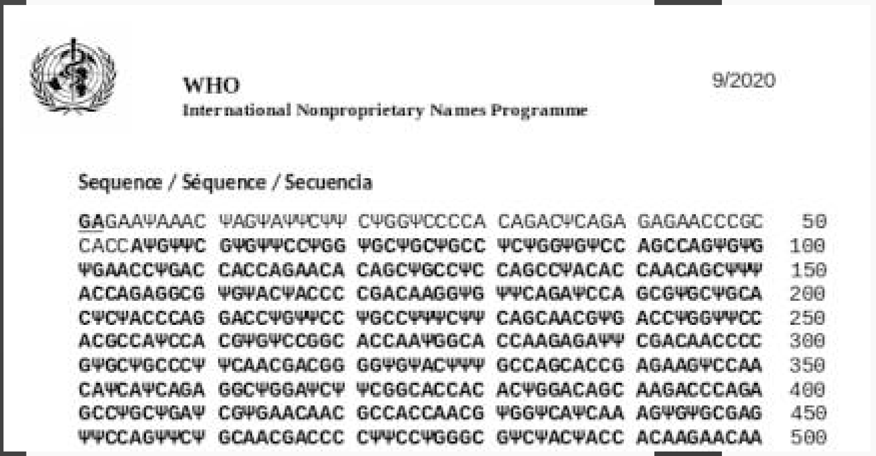
Since I am interested in crispr technology, I am always on the look out for explanations: some articles are very interesting and will provide a lot of better understanding of these very special items and Danny Wind of **the Delphi Company** showed me this one.

Fantastic. Just go there and read it.

Here is given a very good written explanation of things one needs to understand before even being able to follow the article – and it reads like a detective.
Only a small part of the article is published here because all information is on his website and of course much more.

JUST A SHORT INTRODUCTION:
Welcome! In this post, we'll be taking a character-by-character look at the source code of the BioNTech/Pfizer SARS-CoV-2 mRNA vaccine.

This is a good question, so let's start off with a small part of the very source code of the BioNTech/Pfizer vaccine, also known as BNT162b2, also known as Tozinameran also known as Comirnaty.



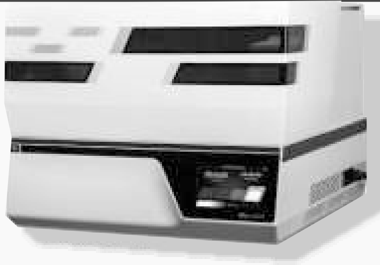
WHO 9/2020
International Nonproprietary Names Programme

Sequence / Séquence / Secuencia

```
GAGAAΨAAAC ΨAGΨAΨΨCΨΨ CΨGGΨCCCCA CAGACΨCAGA GAGAACCCGC    50
CACCAΨGΨΨC GΨGΨΨCCΨGG ΨGCΨGCΨGCC ΨCΨGGΨGΨCC AGCCAGΨGΨG   100
ΨGAACCΨGAC CACCAGAACA CAGCΨGCCΨC CAGCCΨACAC CAACAGCΨΨΨ   150
ACCAGAGGCG ΨGΨACΨACCC CGACAAGGΨG ΨΨCAGAΨCCA GCGΨGCΨGCA   200
CΨCΨACCCAG GACCΨGΨΨCC ΨGCCΨΨΨCΨΨ CAGCAACGΨG ACCΨGGΨΨCC   250
ACGCCAΨCCA CGΨGΨCCGGC ACCAAΨGGCA CCAAGAGAΨΨ CGACAACCCC   300
GΨGCΨGCCCΨ ΨCAACGACGG GGΨGΨACΨΨΨ GCCAGCACCG AGAAGΨCCAA   350
CAΨCAΨCAGA GGCΨGGAΨCΨ ΨCGGCACCAC ACΨGGACAGC AAGACCCAGA   400
GCCΨGCΨGAΨ CGΨGAACAAC GCCACCAACG ΨGGΨCAΨCAA AGΨGΨGCGAG   450
ΨΨCCAGΨΨCΨ GCAACGACCC CΨΨCCΨGGGC GΨCΨACΨACC ACAAGAACAA   500
```

First 500 characters of the BNT162b2 mRNA.
Source: World Health Organization

The BNT162b2 mRNA vaccine has this digital code at its heart. It is 4284 characters long, so it would fit in a bunch of tweets. At the very beginning of the vaccine production process, someone uploaded this code to a DNA printer (yes), which then converted the bytes on disk to actual DNA molecules.



A Codex DNA BioXp 3200 DNA printer

Out of such a machine come tiny amounts of DNA, ending up as RNA in the vaccine vial. RNA is the volatile 'working memory' version of DNA. DNA is like the flash drive storage of biology. DNA is very durable, internally redundant and very reliable. But much like computers do not execute code directly from a flash drive, before something happens, code gets copied to a faster, more versatile yet far more fragile system.
For computers, this is RAM, or biology it is RNA.
The resemblance is striking.
Unlike flash memory, RAM degrades very quickly unless lovingly tended to.

THE BRIEFEST BIT OF BACKGROUND

DNA is a digital code. Unlike computers, which use 0 and 1, life uses A, C, G and U/T (the 'nucleotides', 'nucleosides' or 'bases').
In computers we store the 0 and 1 as the presence or absence of a charge, or as a current, as a magnetic transition, or as a voltage, or as a modulation of a signal, or as a change in reflectivity. Or in short, the 0 and 1 are not some kind of abstract concept.

They live as electrons and in many other physical embodiments.
In nature, A, C, G and U/T are molecules, stored as chains in DNA (or RNA).
In computers, we group 8 bits into a byte, and the byte is the typical unit of data being processed.
Nature groups 3 nucleotides into a codon, and this codon is the typical unit of processing. A codon contains 6 bits of information (2 bits per DNA character, 3 characters = 6 bits. This means 2 = 64 different codon values).
Pretty digital so far. When in doubt, head to the WHO document with the digital code to see for yourself.

SO WHAT DOES THAT CODE DO?
The idea of a vaccine is to teach our immune system how to fight a pathogen, without us actually getting ill. Historically this has been done by injecting a weakened or incapacitated (attenuated) virus, plus an 'adjuvant' to scare our immune system into action. This was a decidedly analogue technique involving billions of eggs (or insects). It also required a lot of luck and loads of time. Sometimes a different (unrelated) virus was also used.
An mRNA vaccine achieves the same thing ('educate our immune system') but in a laser like way. And I mean this in both senses - very narrow but also very powerful.
So here is how it works. The injection contains volatile genetic material that describes the famous SARS-CoV-2 'Spike' protein. Through clever chemical means, the vaccine manages to get this genetic material into some of our cells.
These then dutifully start producing SARS-CoV-2 Spike proteins in large enough quantities that our immune system springs into action. Confronted with Spike proteins, and (importantly) tell-tale signs that cells have been taken over, our immune system develops a powerful response against multiple aspects of the Spike protein AND the production process.
And this is what gets us to the 95% efficient vaccine.

So if your interested: here is his website article:
`https://berthub.eu/articles/posts/reverse-engineering-source-code-of-the-biontech-pfizer-vaccine/`

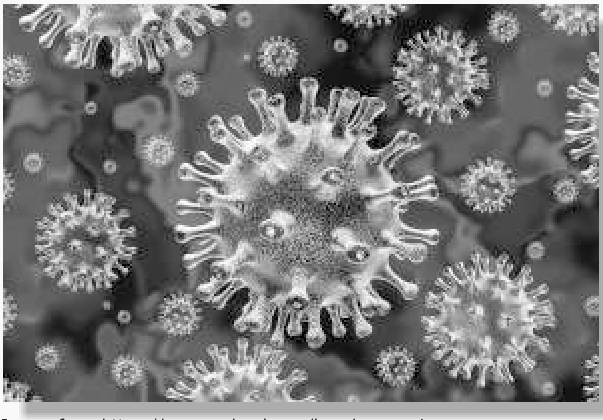


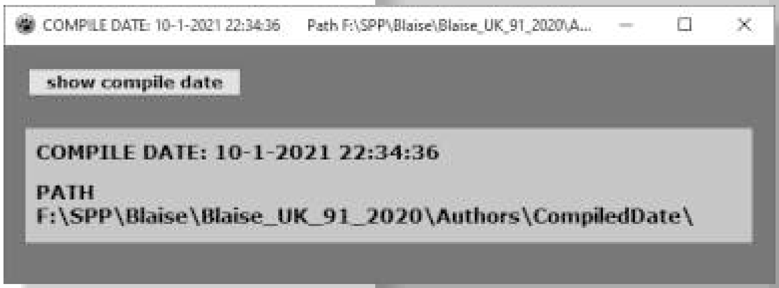
Image from https://www.webmd.com/lung/coronavirus

Further reading/viewing
In 2017 Ben Hubert held a two hour presentation on DNA, which you can view here. **`https://berthub.eu/dna/`**
Like this page it is aimed at computer people.
In addition, Ben Hubert has been maintaining a page on 'DNA for programmers' since 2001. You might also enjoy this introduction to our amazing immune system:
`https://berthub.eu/articles/posts/immune-system/`
Finally, this listing of his blog posts:
`https://berthub.eu/articles/`
has quite some DNA, SARS-CoV-2 and COVID related material.

starter   expert

This is the Lazarus example: I dropped some code that will always show the correct compile-date and path of your application. In contrast to Delphi we need now to create code that is ready for Windows, Linux and Mac
The code is downloadable of course.

```
 unit UnitMain;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Forms, Controls, DateUtils, Graphics, Dialogs, StdCtrls,
  ExtCtrls;

type

  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Panel1: TPanel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private

  public
    Var Compiledate : String; aDate : TDateTime;
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
label1.caption := 'COMPILE DATE:' + ' ' +Compiledate;
label2.caption := 'PATH' + ' '+ ExtractFilePath(Application.ExeName) ; // Path

end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  //You cant use the same solution as in Delphi because
  // we have to serve more oOperating Systems
  if FileAge(Application.ExeName,adate)
  then CompileDate:=DateTimeToStr(aDate)
  else CompileDate:='?';

 Caption := 'COMPILE DATE:' + ' ' +Compiledate +'     '
     + '  Path ' + ExtractFilePath(Application.ExeName) ; // Path

end;
```


COMPILE DATE: 10-1-2021 22:34:36   Path F:\SPP\Blaise\Blaise_UK_91_2020\A...

COMPILE DATE: 10-1-2021 22:34:36

PATH
F:\SPP\Blaise\Blaise_UK_91_2020\Authors\CompiledDate\

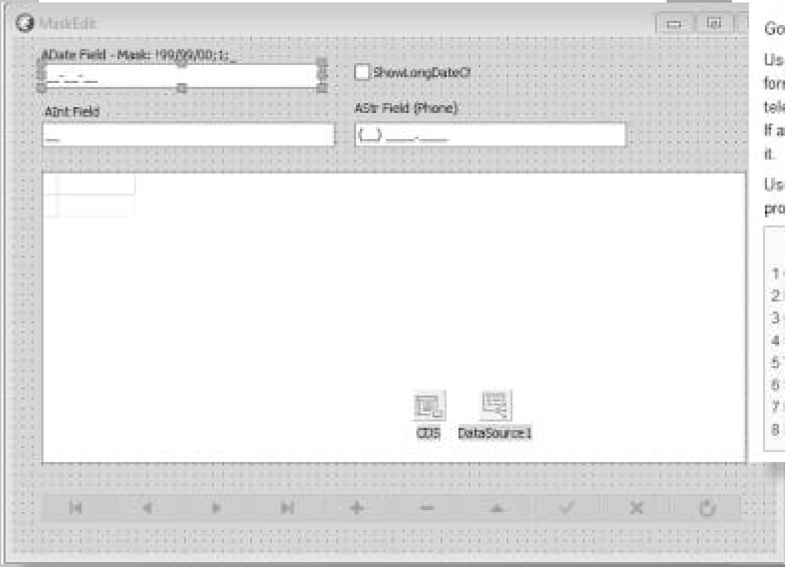
starter expert

This is the promised Delphi version, it has some special aspects which I will show you and that are very determinative: The MaskEdit component has no variant with Database aspects.
So to create almost the same aspects I tried to find a easy way to still get that done.

Introduction:
The “MaskEdit” is a very nice component to use but needs some special knowledge.
In this Delphi example some of the possibilities are explained, the code is available from your personal subscription download address:
`https://www.blaisepascalmagazine.eu/your-downloads/`
and you need to login first.

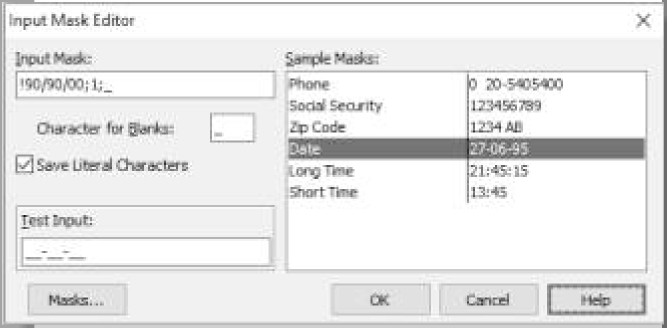
You can build the project easily, its number of components is limited, and you can of course first try out the example. As I mentioned there are some special aspects on this subject. I never realized that there are no standard DBMaskEdit components. So the easy way for the loading and saving in a DataBase must be solved in a work around. Or maybe some special component from third party vendors. But for these easy kind of projects we don’t want that - unless they are free.

The Project
Here is the form of the project:
`FmaskEditForm`



GOALS FOR THE PROJECT:
❶ Explain the settings for the MaskEdit
❷ Creation of the Database
❸ Work around to insert or append
❸ Explain the click event of the GridCell

❶ Explaining the settings for the MaskEdit
The MaskEdit Help is very good at making the process clear:
Choose Edit Mask in the Object Inspector and then click on the ellipsis button:
A new window pops up


Input Mask Editor

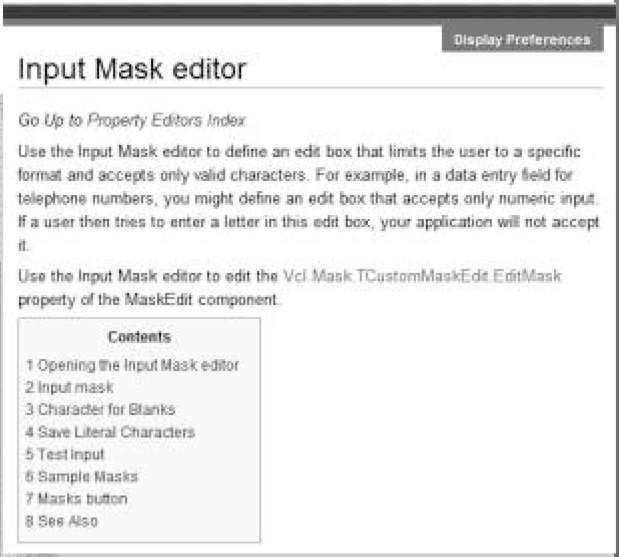
Input Mask: !90/90/00;1;_
Character for Blanks: _
☑ Save Literal Characters
Test Input: __-__-__

Sample Masks:

Phone	0 20-5405400
Social Security	123456789
Zip Code	1234 AB
Date	27-06-95
Long Time	21:45:15
Short Time	13:45

- With that component selected, click the ellipsis button in the Value column for the EditMask property.
- Double-click the Value column for the EditMask property.

If you click on F1 you get these help instructions



Input Mask editor

Go Up to Property Editors Index

Use the Input Mask editor to define an edit box that limits the user to a specific format and accepts only valid characters. For example, in a data entry field for telephone numbers, you might define an edit box that accepts only numeric input. If a user then tries to enter a letter in this edit box, your application will not accept it.

Use the Input Mask editor to edit the Vcl.Mask.TCustomMaskEdit.EditMask property of the MaskEdit component.

Contents
1 Opening the Input Mask editor
2 Input mask
3 Character for Blanks
4 Save Literal Characters
5 Test Input
6 Sample Masks
7 Masks button
8 See Also

Define your own mask:
This is a bit more complex and intricate.
You can use a special character to specify the mask; for a listing of those characters, see the TEditMask datatype. (See figure 2 page before)
The mask consists of three fields separated by semicolons. The three fields are:

❶ The mask itself; you can use predefined masks or create your own.
❷ The character that determines whether or not the literal characters of the mask are saved as part of the data.
❸ The character used to represent a blank in the mask.

I used only standard settings: there is an option to make a localized version:
`FormatSettings.DateSeparator := '/';`
do not use FormatSettings it overules and causes errors.

❷ Creation of the Database

I chose to use a Client data set, because for demo's they are wonderfully simple: her is the code: (*if you want to save the Client database you need to use one of the earlier projects*)

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

  // Use the ShortDateFormat settings only
  // ShowMessage('ddddd = '+ formatdatetime('ddddd', now));
  // Format example : ShowMessage('d/m/yyyy = '+ formatdatetime('d/m/yyyy', now));
  // FormatSettings.DateSeparator := '/';
  // do not use FormatSettings it overules and causes errors

   With CDS do
   begin
    FieldDefs.Clear;

    FieldDefs.Add('ADate', ftDate);
    FieldDefs.Add('AStr',  ftString, 50);
    FieldDefs.Add('AInt',  ftLargeint);
    CreateDataSet;
    //add some data
    Open;

    Append;
    FieldByName('ADate').AsString := '12-09-2003';
    Post;

    Append;
    FieldByName('AInt').AsInteger := 1;
    Post;

    Append;
    FieldByName('ADate').AsString := '12-12-2090';
    FieldByName('AInt').AsInteger := 30;
    Post;
    DBGrid1.Columns[0].Width := 150;  // this sets the date colum width
                  //which is otherwise to small for the long date
  end ;

end;
```

So this creates each time the database and fills it with fields and data.
The event should be `OnCreate of the form`

❸ Work around to insert or append

We need to be able to extract as well to insert dat from the database (grid). To be able to insert I thought it might be the easiest way to use the MaskEdit because we then do not have to use extra components, but there is a restriction: the `onClick` is triggerd directly, so I used the `onDoubleclick` event. `OnChange` has the same problem. Here is an example:

```
procedure TFMaskEdit.MaskEdit1DblClick(Sender: TObject);
begin // Must be doubleclick otherwise it will be triggerd to early
  // sets the value in the database and shows it in the grid
  With CDS do
   begin
    Append;
    FieldByName('ADate').AsDateTime :=
            StrToDate(MaskEdit1.EditText);
    Post;
   end;
end;
```

❹ Explain the click event of the GridCell
I got almoast frustrated when I tried to implement whta i thought it was a simple
Nothing hapne nothing worked
ctivate a simple gridcel.

e the way TDBGrid is coded,
dataset is synchronized to the
ed/clicked grid row. Generally
siest to get values from the
f the dataset, but you asked,
changing the current record's
ulating the cell's text because
fight you every inch of the way.

e robust way of getting the cell
udes Remy Lebeau's suggestion
ield instead of SelectedField, is

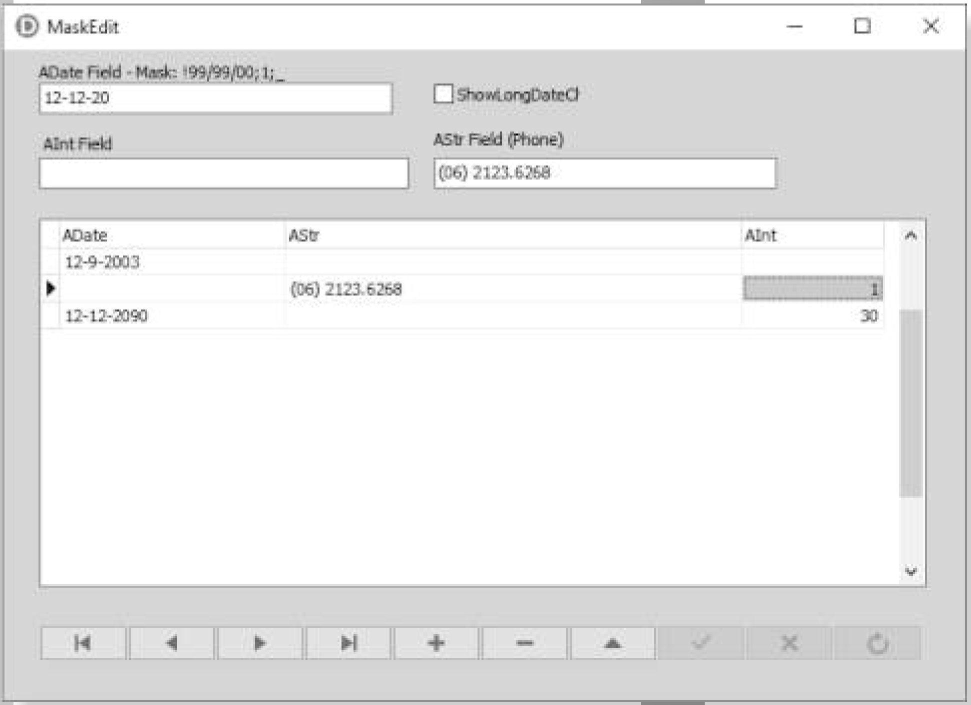
❹ Explain the click event of the GridCell

I got almost frustrated when I tried to implement what I thought was a simple peace of code.
Nothing happened, nothing worked as I wanted to activate a simple gridcell. I found a solution that solves this problem on the internet and explains it as follows:

It works because the way TDBGrid is coded, the associated dataset is synchronized to the currently selected/ clicked grid row.
Generally speaking, it's easiest to get values from the current record of the dataset.
Try to avoid changing the current record's values by manipulating the cell's text because the DBGrid will fight you every inch of the way.
Note that a more robust way of getting the cell text, which includes **Remy Lebeau's** **`(http://www.lebeausoftware.org)`** suggestion is to use `Column.Field` instead of `SelectedField`, see code right top:

```
procedure TFMaskEdit.DBGrid1ColEnter(Sender: TObject);
begin
 // Starts the field value whre you cliked on immediately
 MaskEdit2.EditText := cds.FieldByName('Aint').Asstring;
end;

procedure TFMaskEdit.DBGrid1CellClick(Column: TColumn);
var s : string;
  AField : TField;
begin
 // Starts the column field value
 // if you do not use them together you will mis the first instance of Cell Click
 AField := column.field ; // Fill the field
 S:= AField.AsString;    // Fil the Var

 MaskEdit2.EditText := S; // Set into Maskedit
 // it must be MaskEdit2.EditText, not MaskEdit2.EditMask!

end;
```

I chose to use **`DBGrid1colEnter`** for initializing the event, and the **`DBGrid1CellClick`** to fill te the MaskEdit: it must be **`MaskEdit2.EditText`**, not **`MaskEdit2.EditMask`**!
So now you can click on the gridcell and see the changes....



12-12-20

(06) 2123.6268

ADate	AStr	AInt
12-9-2003		
	(06) 2123.6268	1
12-12-2090		30

starter expert

In this series we want to show and create little nice code creations that can be helpful and very easy to replicate.

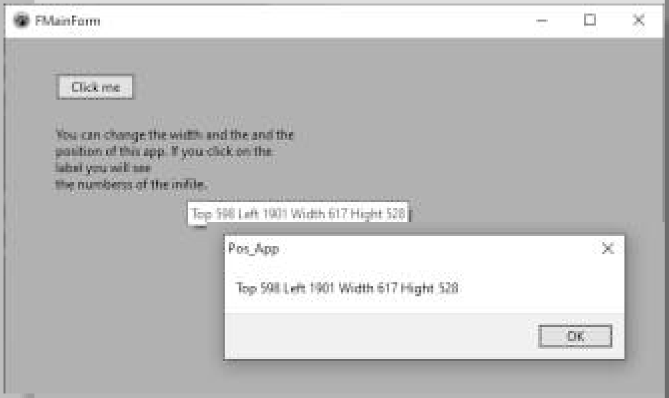
INTRODUCTION:
As I promised: here is the version for Lazarus. This one can be used on Windows / Linux and Mac OS (Big Sur).
Mattias Gaertner whom fell in to the bucket of Pascal-Elixir (I am jealous of this) has of course helped me with Mac OS.
I had learned by one of our readers that if you want to write an app for several Oss you best start with Mac. Usually after that Linux and Windows will be easy
This app is created with the latest version of Lazarus 2.0.10 / FPC 3.2.0.
I have tested it under all OS's.

It's a very simple project as you can see by the code. I added some extra's to show the actual number if you change the size and or placement on the desktop.

We build in a procedure to make sure that the Application-window-sizes are guarded.
(*See the code at the red arrow on the next page.*)
I had to do this because I found out - I am normally on a very large 4k screen - under "Big Sur" the application flipped and was out of sight. That is because my "Big Sure" screen is only 13 inches.

Actually there are only two important events: Form Create and Form Close.
The `TMemInifile.Create(extractfilepath (application.exename) +'PositionIni.ini');` means that if the file does not exist it will be created and the position of the form as well its size will be read.
In the Form Close it works the other way around, the settings will be rewritten into the inifile.
At that point the inifile must be freed: `PositionIni.Free;` In case you are using several forms you should free in the FormDestroy of the main form.

To demonstrate, I made a small text which you can activate by clicking the button. It will show the text and if you click on the text you will see a `showmessage` that lets you see the numbers of the position and the size of the app.
After that there is a read out inserted into the Hint. If you move over with your mouse it will show up. If you change the position and/or size you will after clicking on the text again be able to see the changes. You can see that on the next page... as an extra I have some information over Mac and the Bundles. Read it: for Mac it is important...

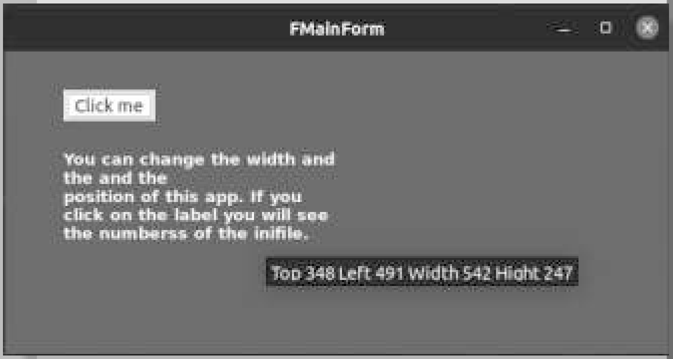


You can change the width and the and the
position of this app. If you click on the
label you will see
the numberss of the inifile.

Top 598 Left 1901 Width 617 Hight 528

Top 598 Left 1901 Width 617 Hight 528

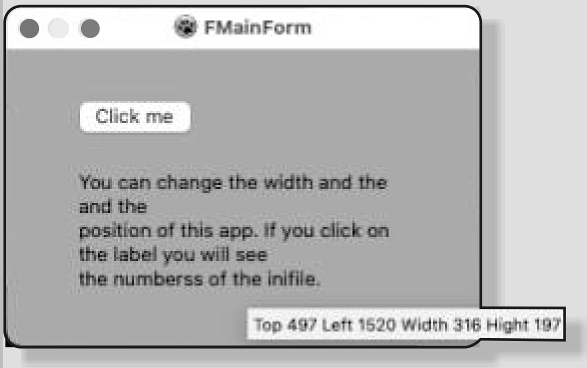
Figure 1: Push the button , Windows 10 example



You can change the width and
the and the
position of this app. If you
click on the label you will see
the numberss of the inifile.

Top 348 Left 491 Width 542 Hight 247

Figure 2: The Hint appears, Ubuntu Linux example



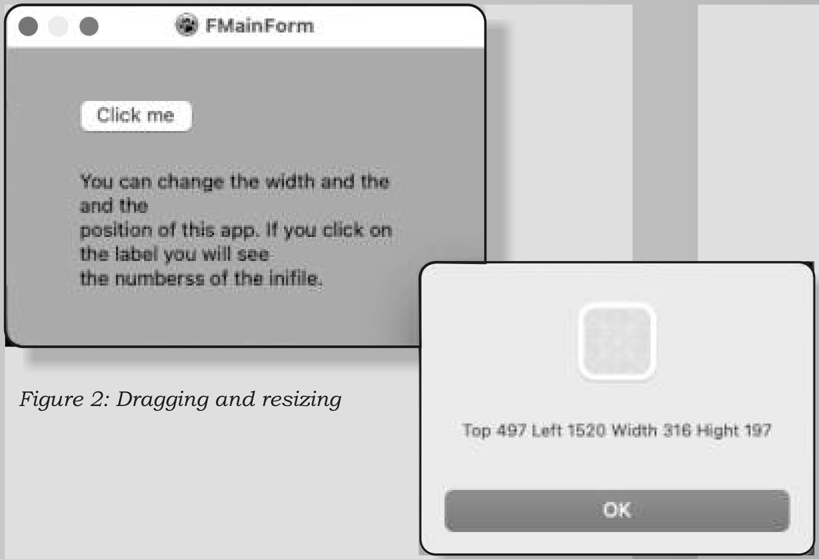
You can change the width and the
and the
position of this app. If you click on
the label you will see
the numberss of the inifile.

Top 497 Left 1520 Width 316 Hight 197

Figure 3: Osx Mac, "Big Sur"

```
 unit fPositionAppMainForm;

{$mode objfpc}{$H+}

interface

uses
 Classes, SysUtils, Forms, Controls, Graphics, Dialogs, StdCtrls, Inifiles,
 LazLogger{For testing Purpose} ;

...

 var
 FMainForm: TFMainForm;

implementation

{$R *.lfm}

function TFMainForm.GetCfgFilename: string;
{$IFDEF darwin}
var
 p: SizeInt;
{$ENDIF}
begin
 Result:=ExtractFilePath(Application.ExeName);
 {$IFDEF darwin}
 p:=pos('.app/Contents/MacOS/',Result);
 if p>0 then
  Result:=ExtractFilePath(LeftStr(Result,p-1));
 {$ENDIF}
 Result:=Result+'PositionIni.ini';
end;

 { TFMainForm }
procedure TFMainForm.FormCreate(Sender: TObject);
Var  r: TRect;
begin
 // PositionIni    := TMemInifile.Create(extractfilepath(application.exename) +'PositionIni.ini');
 // this is the standard windows solution

 // These lines below show the construction you need
 // if you want to use it on Mac, Linux and Windows
 PositionIni    := TMemInifile.Create(GetCfgFilename);

 PositionTop    := PositionIni.ReadInteger('SETTINGS', 'Top', PositionTop) ;
 PositionLeft   := PositionIni.ReadInteger('SETTINGS', 'Left',PositionLeft) ;

 AppWidth       := PositionIni.ReadInteger('SETTINGS', 'Width', AppWidth) ;
 AppHeight      := PositionIni.ReadInteger('SETTINGS', 'Height', AppHeight) ;

 // To test the eventual wrong numbers this is a way to ensure
 // that it will only start with the maximum width and higth
 r:=Screen.WorkAreaRect;
 if (PositionLeft<r.Left) or (PositionTop<r.Top)
   or (PositionLeft+AppWidth>r.Right)
   or (PositionTop+AppHeight>r.Bottom) then
 begin
  // the actual test itself:
  // outside of screen, using default
  //ShowMessage('AAA1 '+dbgs(PositionLeft)+' '+dbgs(r));
 end else
  FMainForm.SetBounds(PositionLeft,PositionTop,AppWidth,AppHeight);
  //ShowMessage('AAA2 '+dbgs(PositionLeft)+' '+dbgs(r)+' Cur='+dbgs(BoundsRect));


 // to set the form at the right placee
 FMainForm.Top   := PositionTop;
 FMainForm.Left  := PositionLeft;
 FMainForm.Width := AppWidth;
 FMainForm.Height := AppHeight;
end;
```

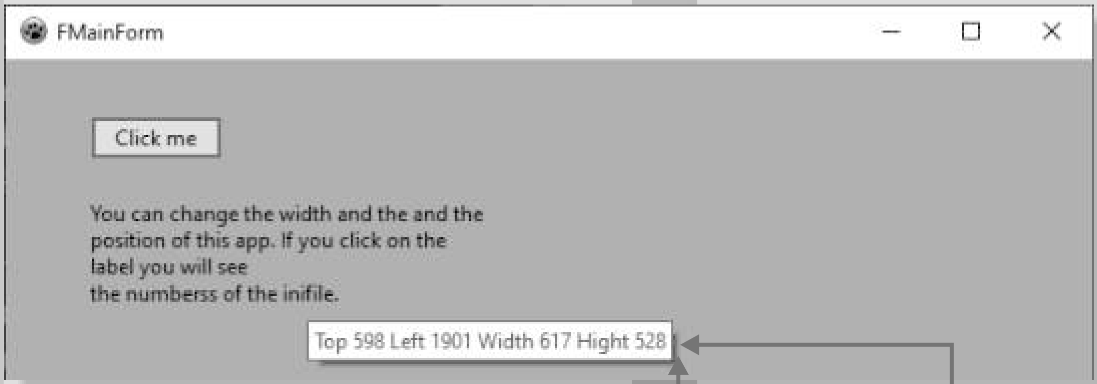
```
procedure TFMainForm.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  PositionTop  := FMainForm.Top;
  PositionLeft := FMainForm.Left;

  AppWidth     := FMainForm.Width;
  AppHeight    := FMainForm.Height;

  PositionIni.WriteInteger('SETTINGS', 'Top',  PositionTop);
  PositionIni.WriteInteger('SETTINGS', 'Left', PositionLeft);

  PositionIni.WriteInteger('SETTINGS', 'Width', AppWidth);
  PositionIni.WriteInteger('SETTINGS', 'Height', AppHeight);

  PositionIni.UpdateFile;
  PositionIni.Free ;
end;

procedure TFMainForm.Button1Click(Sender: TObject);
begin
   // popup the label
  Label1.Visible := True ;
end;

procedure TFMainForm.FormDestroy(Sender: TObject);
begin
  // PositionIni.free; //Only if there are several forms
end;


procedure TFMainForm.Label1Click(Sender: TObject);
Var PT,Pl,AW,AH : String;
  Hintstring: String;

begin  // create the Hintstring and the showmessage

  PositionTop    := FMainForm.Top;
  PositionLeft   := FMainForm.Left;

  AppWidth       := FMainForm.Width;
  AppHeight      := FMainForm.Height;

  PositionIni.WriteInteger('SETTINGS', 'Top',  PositionTop);
  PositionIni.WriteInteger('SETTINGS', 'Left', PositionLeft);

   PositionIni.WriteInteger('SETTINGS', 'Width', AppWidth);
  PositionIni.WriteInteger('SETTINGS', 'Height', AppHeight);

  PositionIni.UpdateFile;
  PositionTop    := PositionIni.ReadInteger('SETTINGS', 'Top', PositionTop) ;
  PositionLeft   := PositionIni.ReadInteger('SETTINGS', 'Left',PositionLeft) ;

  AppWidth       := PositionIni.ReadInteger('SETTINGS', 'Width', AppWidth) ;
  AppHeight      := PositionIni.ReadInteger('SETTINGS', 'Height', AppHeight) ;

  PT  := IntTostr(PositionTop);
   PL  := IntTostr(PositionLeft);
  AW  := IntTostr(AppWidth);
  AH  := IntTostr(AppHeight);

  Hintstring := 'Top' + ' ' + PT + ' ' + 'Left' + ' ' + PL + ' ' + 'Width' + ' ' + AW + ' '+'Hight' + ' ' + AH;

  Label1.Hint:= Hintstring;

  showmessage(Hintstring);

end;
```



You can change the width and the
and the
position of this app. If you click on
the label you will see
the numberss of the inifile.

Top 497 Left 1520 Width 316 Hight 197

Figure 2: Dragging and resizing



You can change the width and the and the
position of this app. If you click on the
label you will see
the numberss of the inifile.

Top 598 Left 1901 Width 617 Hight 528

Figure 3: Now the hint appears

```
procedure TFMainForm.Label1Click(Sender: TObject);
Var PT,Pl,AW,AH : String;  Hintstring: String;
begin
 PositionTop := FMainForm.Top;
 PositionLeft:= FMainForm.Left;
 AppWidth   := FMainForm.Width;
 AppHeight  := FMainForm.Height;

 PositionIni.WriteInteger('SETTINGS', 'Top',  PositionTop);
 PositionIni.WriteInteger('SETTINGS', 'Left', PositionLeft);
 PositionIni.WriteInteger('SETTINGS', 'Width', AppWidth);
 PositionIni.WriteInteger('SETTINGS', 'Height', AppHeight);

 PositionIni.UpdateFile;
 PositionTop  := PositionIni.ReadInteger('SETTINGS', 'Top', PositionTop) ;
 PositionLeft  := PositionIni.ReadInteger('SETTINGS', 'Left',PositionLeft) ;

 AppWidth    := PositionIni.ReadInteger('SETTINGS', 'Width', AppWidth) ;
 AppHeight    := PositionIni.ReadInteger('SETTINGS', 'Height', AppHeight) ;
 PT :=  IntTostr(PositionTop);
 PL :=  IntTostr(PositionLeft);
 AW :=  IntTostr(AppWidth);
 AH :=  IntTostr(AppHeight);

 Hintstring := 'Top' + ' ' + PT + ' ' + 'Left' + ' ' + PL + ' ' + 'Width' + ' ' + AW + ' '+'Hight' + ' ' + AH;
 Label1.Hint:= Hintstring;
 showmessage(Hintstring);
end;
```

LAZARUS INSTALL FOR PAS2JS

By Mattias Gärtner starter expert

INTRODUCTION

COMPILER

Pas2js is an open source Pascal to JavaScript transpiler.
It parses Object Pascal and emits JavaScript. The JavaScript is currently of level ECMAScript 5 and should run in any browser or in Node.js (target "nodejs"). It is available in 3 forms:

- as a library
- as a command-line program
- as a webserver

It transpiles from actual Pascal source, it has no intermediate `.ppu` files.
That means all sources must always be available.
Through external class definitions, the compiler can use JavaScript classes:

- All classes available in the JavaScript runtime, and in the browser are available through import units (comparable to the windows or unix units for the native compiler).
- For Node.js, basic support for the nodejs runtime environment is available.
- An import unit for jQuery is available (libjquery)

This project is **NOT** related to a similar named project on github.

WIKIPEDIA *A source-to-source translator, source-to-source compiler (S2S compiler), transcompiler, or* ***transpiler*** *is a type of translator that takes the source code of a program written in a programming language as its input and produces an equivalent source code in the same or a different programming language.*

A ***transpiler*** *converts between programming languages that operate at approximately the same level of abstraction, while a traditional compiler translates from a higher level programming language to a lower level programming language.*

For example, a ***transpiler*** *may perform a translation of a program from* ***Pascal to JavaScript****, while a traditional compiler translates from a language like C to Assembler or Java to* ***bytecode.***

Bytecode, *also termed portable code or p-code, is a form of instruction set designed for efficient execution by a software interpreter.*
Unlike human-readable source code, bytecodes are compact numeric codes, *constants, and references (normally numeric addresses) that encode the result of compiler parsing and performing semantic analysis of things like type, scope, and nesting depths of program objects.*
The name bytecode stems from instruction sets that have one-byte opcodes *(is the portion of a machine language instruction that specifies the operation to be performed.) followed by optional parameters.*

RTL
For the generated code to work, a small JavaScript file is needed: `rtl.js.` It defines an object rtl. This object will start the Object Pascal code if you include a call to `rtl.run()` in the HTML page.

```
<script
type="application/javascript">
  rtl.run()
</script>
```

pas2js can automatically include this file in the generated output, like this:

```
pas2js -Jc -Jirtl.js -Tbrowser
hello.pas
```

For nodejs, the compiler will insert the call to `rtl.run()` automatically at the end of the generated Javascript file.

There is a basic **Object Pascal RTL**, several units from the FPC Packages are also available
For extra information you can go to:
`https://wiki.freepascal.org/pas2js`

The most recent version you can get from our own download addresses:
`https://www.blaisepascalmagazine.eu/pas2js-version2/` where you find:

```
pas2js-windows-2.0.0.zip
pas2js-macos-2.0.0.zip
pas2js-linux-2.0.0.zip
```

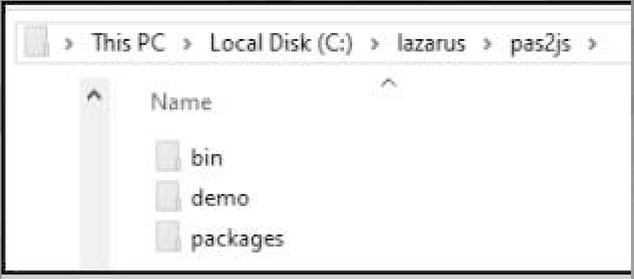
or you can find it on this webaddress where all and older versions are available:
`ftp://ftpmaster.freepascal.org/fpc/contrib/pas2js`
The releases contain binaries for Windows(32 and 64bit), Linux (64 bit) and macOS.

Installation procedure:

1. Download pas2js
   Every version has a directory with the version number.
   A list of changes can be found on the changelog page Pas2JS Version Changes
2. Unpack it in folder of your choice.
   The example top right uses `C:\lazarus\pas2js\`.
   The release contains three folders:
   - **`bin`**
     - contains the compiler as executable **`(pas2js`** or **`pas2js.exe)`** and library and some utilities.
   - **`demo`**
     - lots of examples
   - **`packages`**
     - the Pascal units of the RTL and other packages.
3. You can create a simple config to let the compiler find the RTL and packages.

Edit **`bin/pas2js.cfg`**:

```
#
# Minimal config file for pas2js compiler
#
# -d is the same as #DEFINE
# -u is the same as #UNDEF
#
# Write always a nice logo ;)
-l

# Display Hints, Warnings and Notes
-vwnh
# If you don't want so much verbosity use
#-vw

# Allow C-operators
-Sc

-Fu$CfgDir\..\packages\*
-Fu$CfgDir\..\compiler\utils\pas2js\dist

#IFDEF nodejs
-Jirtl.js
#ENDIF

# end.
```



SVN

```
svn co
https://svn.freepascal.org/svn/
projects/pas2js/trunk pas2js
```

You need FPC 3.0.4 or better to compile it.

Change to the directory and build it with:

```
make clean all
```

This creates
`bin/$(TargetCPU)-$(TargetOS)/pas2js` **`(Windows: pas2js.exe)`**.
For example on Linux 64bit it creates **`bin/x86_64-linux/pas2js`**,
while under Windows 64bit it creates bin\x86_64-win\pas2js.exe.

And create a text file **`pas2js.cfg`** in the folder where **`pas2js.exe`** is:

```
# Write always a nice logo ;)
-l

# Display Warnings, Notes and Hints
-vwnh
# If you don't want so much verbosity
use
#-vw

-Fu$CfgDir/../../packages/*
-Fu$CfgDir/../../compiler/utils/pas2j
s/dist/

#IFDEF nodejs
-Jirtl.js
#ENDIF

# Put all generated JavaScript into
one file
-Jc

# end.
```

HOW TO USE PAS2JS

The command-line arguments are kept mostly the same as the FPC command-line arguments. Error messages are also in the same format.

The compiler needs access to all sources, and so you need to specify the path to the sources of all used units.

As for the FPC compiler, a configuration file is supported, which has the same syntax as the FPC config file. Note that the snapshots and svn version already contains a default pas2js.cfg with unit search paths (-Fu) for the rtl and fcl. See here how for details about the pas2js.cfg.

Basically, the command is the same as any FPC command line. The only thing that is different is the target: -Tbrowser or -Tnodeejs

Here is the complete list of command line arguments.

FOR THE BROWSER

Consider the classical:

```
program hello;

begin
 Writeln('Hello, world!');
end.
```

Yes, writeln is supported.
Here is how to compile it:

```
pas2js -Jc -Jirtl.js -Tbrowser hello.pas
```

When compiled succesfully, the code can be run in the browser by opening a html file in the browser with the following content:

```
<html>
  <head>
    <meta charset="utf-8"/>
    <script type="application/javascript"
src="hello.js"></script>
  </head>
  <body>
    <script type="application/javascript">
     rtl.run();
    </script>
  </body>
</html>
```

The files that are needed are:
hello.html
hello.js
Whether hello.html is opened by double-clicking it in the explorer or put on a server and opened with an URL, is not relevant for the functioning.
The output is visible in the browser's web developer console. By including the browserconsole unit, it will be visible in the browser page:

```
program hello;

uses browserconsole;

begin
 Writeln('Hello, world!');
end.
```

FOR NODEJS

```
pas2js -Tnodejs hello.pas
```

When compiled succesfully, the code can be run in node using the following command.

```
nodejs hello.js
```

Note: on macOS it is "node hello.js"

What is node js used for? Node. js is primarily used for non-blocking, event-driven servers, due to its single-threaded nature. It's used for traditional web sites and back-end API services, but was designed with real-time, push-based architectures in mind.

WIKIPEDIA *Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.*

Though .js is the standard filename extension for JavaScript code, the name "Node.js" doesn't refer to a particular file in this context and is merely the name of the product. Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

The Node.js distributed development project was previously governed by the Node.js Foundation, and has now merged with the JS Foundation to form the OpenJS Foundation, which is facilitated by the Linux Foundation's Collaborative Projects program.

LAZARUS INTEGRATION OF PAS2JS

Lazarus understands the concept of external classes as used by pas2js, so code completion will work. Since Lazarus 1.9 the IDE can use **`pas2js.exe`** as a normal compiler.

The integration is described on the Lazarus website: lazarus -pas2js integration. It is still under construction, but deep integration with lazarus is planned.

Importing Javascript classes
To import a javascript class, one writes a normal class definition that mimics the Javascript class. It is possible to use properties. Many examples can be found in the JS, web, nodejs and libjquery units.

```
TJSFunction = class external name 'Function'(TJSObject)
private
 Flength: NativeInt external name 'length';
 Fprototyp: TJSFunction external name 'prototyp';
public
 name: String;
 property prototyp: TJSFunction read Fprototyp;
 property length: NativeInt read Flength;
 function apply(thisArg: TJSObject; const ArgArray: TJSValueDynArray): JSValue; varargs;
 function bind(thisArg: TJSObject): JSValue; varargs;
 function call(thisArg: TJSObject): JSValue; varargs;
end;
```

This declares the TJSFunction object: in Javascript, functions are objects.

The "external name 'Function'" means that you declare a Javascript class where the Javascript name of the class is 'Function'.
The (TJSObject) means it descends from TJSObject also an external class. There does not need to be an ancestor type.
Fields are declared just as in Pascal.
To declare read-only fields, a trick can be used: declare the field using an external name "thename" modifier, and declare a read-only property with the same name.
(see the length declaration)
Varargs can be used to indicate that a function accepts any number of arguments.
JSValue can be used to indicate an unknown type.
It is more or less equivalent to a Variant.

Create simple JS objects with the new function
Some JS-framework functions expect a JS object as parameter. Here is how to do that in Pascal using the new function from unit JS:

```
// JavaScript:
DoIt({name:"Fred", id:3, size:4.3});
// Pascal;
DoIt(new(['name','Fred', 'id',3, 'size',4.3]));
You can nest it to create sub objects:

// JavaScript:
DoIt({name:"Fred", size:{width:3,height:2}});
// Pascal;
DoIt(new(['name','Fred', 'size',new(['width',3, 'height',2])]));
You can use TJSArray._of to create JS arrays on the fly

// JavaScript:
DoIt({numbers:[1,2,3]});
// Pascal;
DoIt(new(['numbers',TJSArray._of(1,2,3)]));
```

RESOURCE STRINGS

The pas2js transpiler can generate a JSON file (extension .jrs) with all the resource strings in your program.

This is a quite simple file. A JSON object exists for every unit, with each JSON property a resource string.

```
{
 "trs2" : {
   "ResUsed" : "This resourcestring is used",
   "ResUnUsed" : "This resourcestring is not used",
   "ImplResUsed" :
   "This implementation resourcestring is used"
 }
 "trs1" : {
   "MyString" : "The very nice string we will need to
translate"
 }
```

This file can be translated, and the translation file can be loaded using the rstranslate unit, part of the rtl. There are demo programs which demonstrate the use of this feature.

The generating of this file is controlled by the -Jr option. It can take 3 possible arguments:

❶ `none` This is the default, no file is generated.
❷ `unit` one file per compiled unit will be generated. This file will contain all resource strings of the unit.
❸ `program` one file is generated for the main file. This fill will contain all used resource strings for the main file and all the units it uses.

If you compile a program, then the program option will generate a file with all the used resource strings in your program.

The above example was generated using the command:

```
pas2js -Jrprogram trs1.pp -B
```

Note that the format is different from the format used by **FPC:**

Identifiers in the file are case sensitive: the names must be typed as they appear in the source file. The strings are grouped per unit, this allows to load them faster.
The hash and bytes parts are missing, they make little sense in a Javascript context.

Exceptions
Exceptions are translated to actual Javascript exceptions. The `rtl.js` has several mechanisms to deal with uncaught exceptions. The basic mechanism is setting the `showUncaughtExceptions` to true before calling `rtl.run()` in your html file.

```
<script
type="application/javascript">
  rtl.showUncaughtExceptions=true;
  rtl.run();
</script>
```

The browser will then use a `window.alert()` to show uncaught exceptions.
More explanations can be found in `pas2js_exceptions`

DEBUGGING
The generated Javascript source code is of course visible and debuggable in the browser.

Moreover, the transpiler can generate a source map, which means that you will be able to see and debug the Pascal code in the browser. (*Not everything will work, but many things do. This depends on the browser too.*)

A source map can be generated using the command-line parameter:

```
-Jm
```

The easiest is to include the Pascal sources in the source map:

```
-Jminclude
```

By default all source filenames are relative to `.js.map`. You can tell the compiler to store all file names relative to a specific local base directory:

```
-Jmbasedir=DirName
```

And you can store an URL in the map, so the browser will use URL/above-relative-file-name to get the source:

```
-Jmsourceroot=URL
```

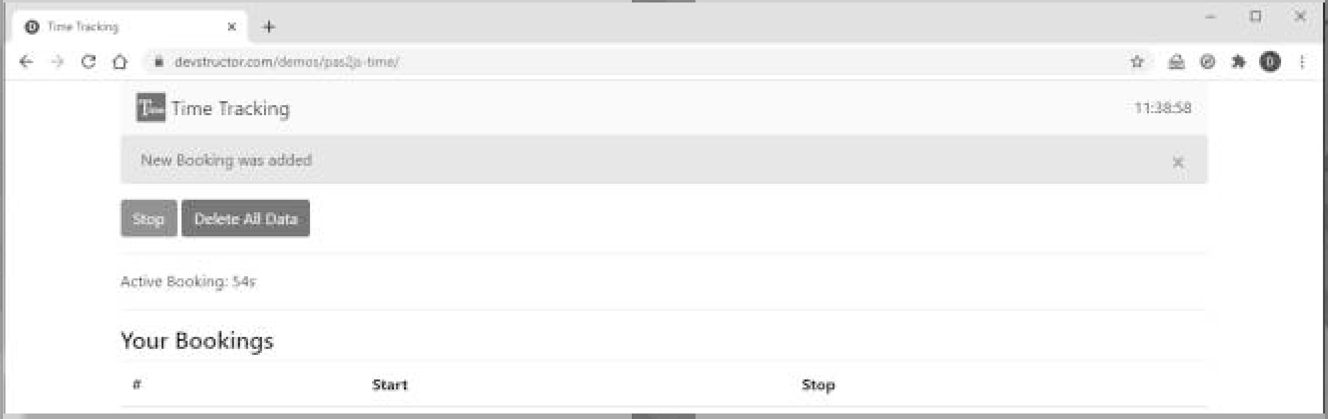
Porting from FPC/Delphi
See here for tips and traps porting code from FPC and Delphi.

Delphi cannot parse some of the constructs that exist in pas2js (namely: external classes). You can create stub declarations suitable for the Delphi parser with the stub creator.

EXAMPLES:

- Time Tracking Application:
  `https://www.devstructor.com/demos/pas2js-time/source.zip`
- Drawing and animation on canvas:
  `http://ragnemalm.se/images/santa/santa.html`
  (sources: `http://ragnemalm.se/images/santa/`)
- WebGL:
  `https://github.com/genericptr/Pas2JS-WebGL#pas2js-webgl`
- Allegro Web Game:
  `https://lainz.github.io/AllegroPas2JS-Demo-Game/index.html`
  (sources: `https://github.com/lainz/AllegroPas2JS-Demo-Game`)
  `https://github.com/genericptr/Pas2JS-WebGL#pas2js-webg`

In the next issue we will explain the Lazarus pas2JS Integration

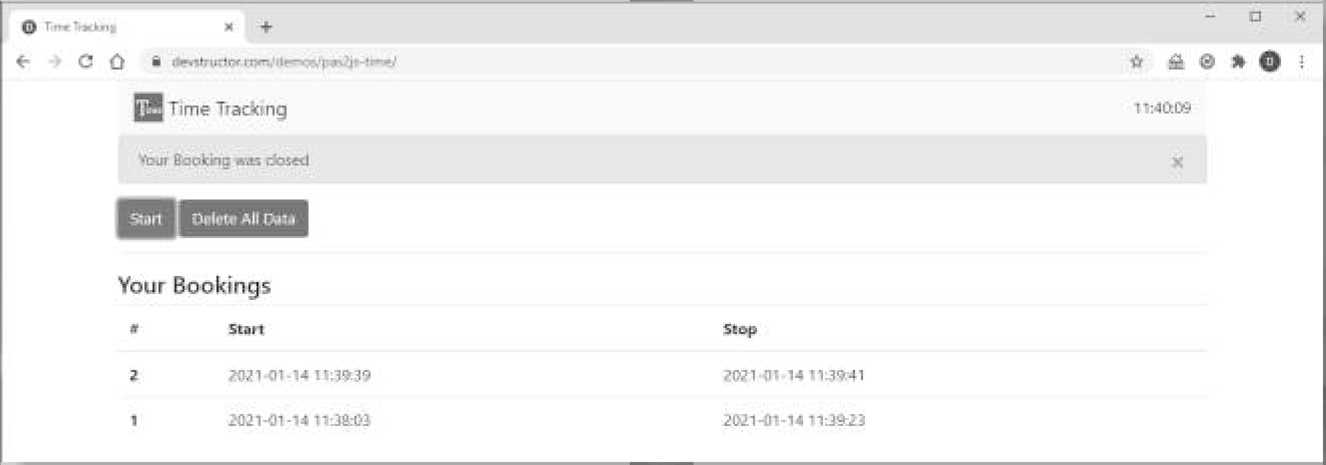

Time Tracking 11:38:58

New Booking was added

Active Booking: 54s

Your Bookings

#	Start	Stop

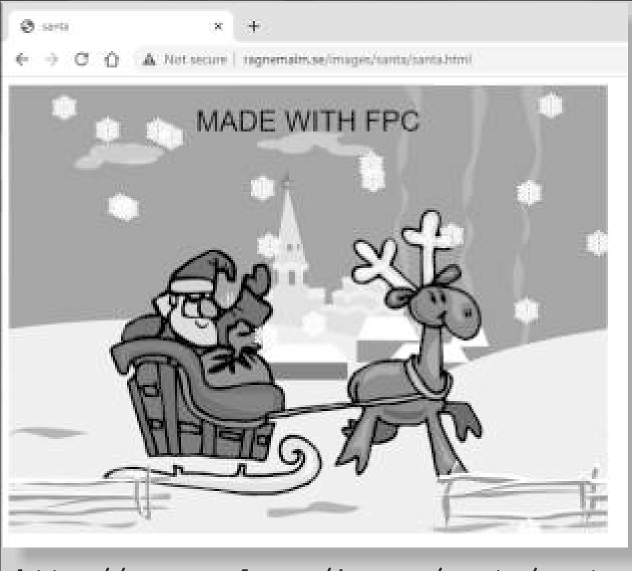

Time Tracking 11:40:09

Your Booking was closed

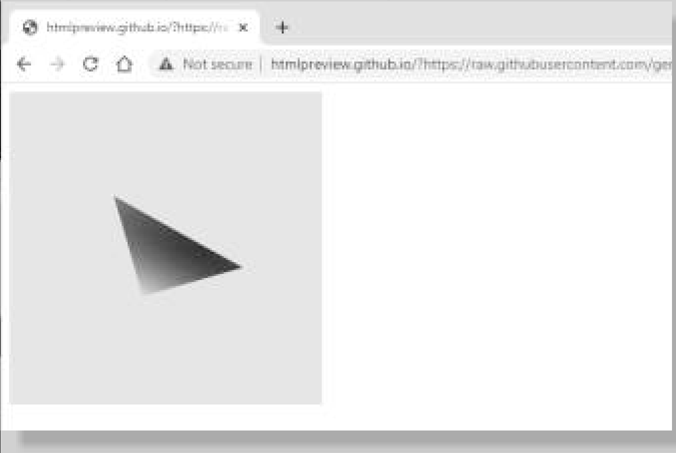
Your Bookings

#	Start	Stop
2	2021-01-14 11:39:39	2021-01-14 11:39:41
1	2021-01-14 11:38:03	2021-01-14 11:39:23

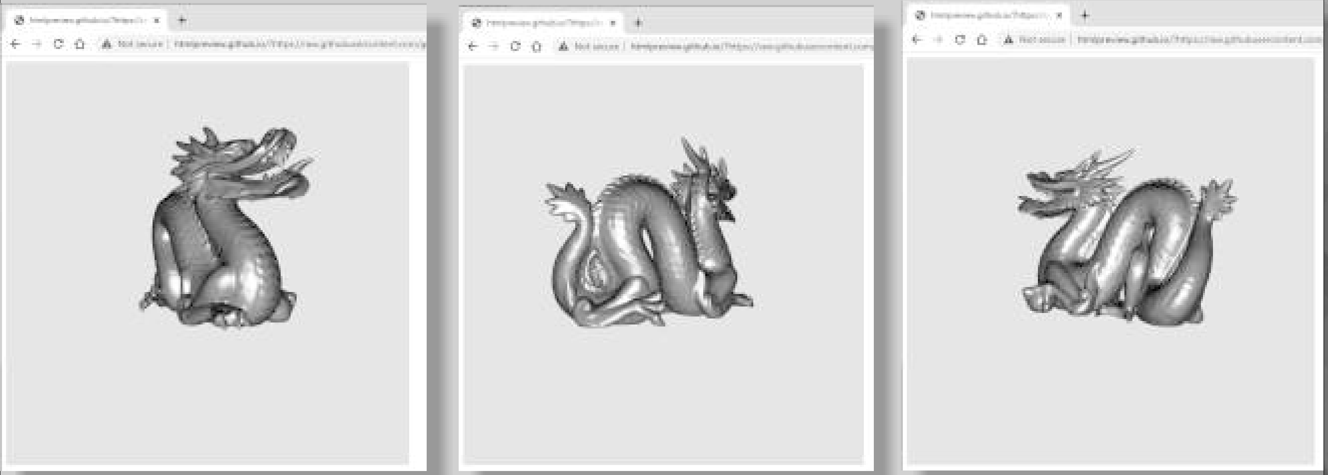
Figure 1+2: Time Tracking Application:



http://ragnemalm.se/images/santa/santa.html

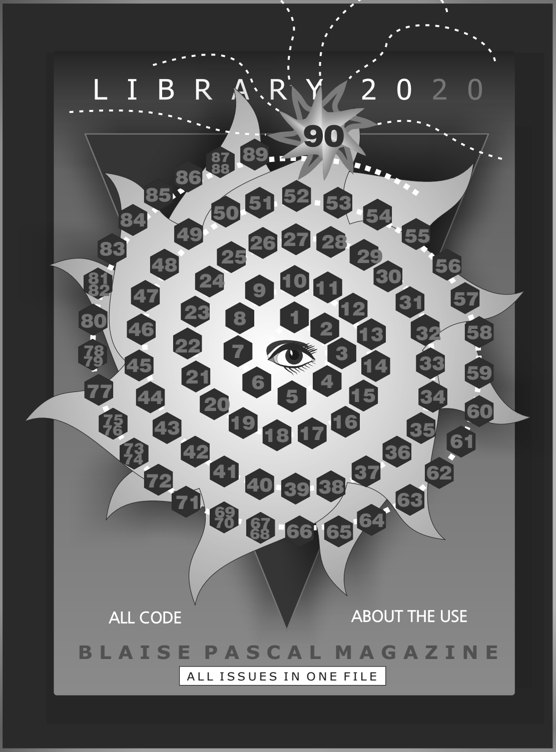

htmlpreview.github.io/?https://
Not secure | htmlpreview.github.io/?https://raw.githubusercontent.com/ger

https://github.com/genericptr/Pas2JS-WebGL#pas2js-webgl

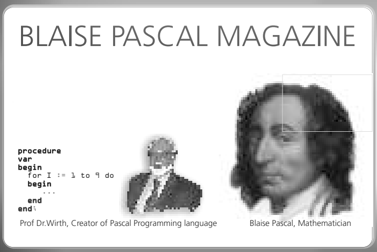


https://lainz.github.io/AllegroPas2JS-Demo-Game/index.html

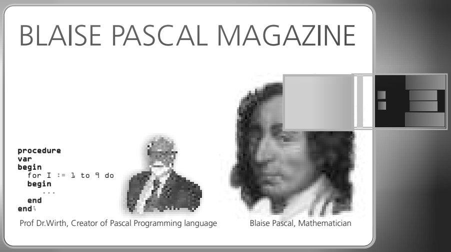
ADVERTISEMENT



BLAISE PASCAL MAGAZINE 90
Object Pascal / Internet / JavaScript / WebAssembly / Pas2Js / Databases
CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux
Blaise Pascal
Object Pascal Handbook for Delphi 10.4 Sydney Edition / By Marco Cantu
PPM Viewer for CNN feature maps / By Max Kleiner
The latest FastReport is also available for Lazarus / By Detlef Overbeek
MindMap: the one that is simply the absolute top / By Detlef Overbeek
Database Workbench 5.7 you simply cant do without it / By detlef Overbeek
Code Examples part 3/4/5
Free webcore for the Mac / By Mattias Gaertner
MMXcode for Delphi / By Detlef Overbeek
Free Simple projects with Lazarus and Delphi
Cyclic redundancy codes / By David Dirkse


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

1 year Subscription+
the new LibStick (6)
(on USB Card - 90 Issues)

€ 100

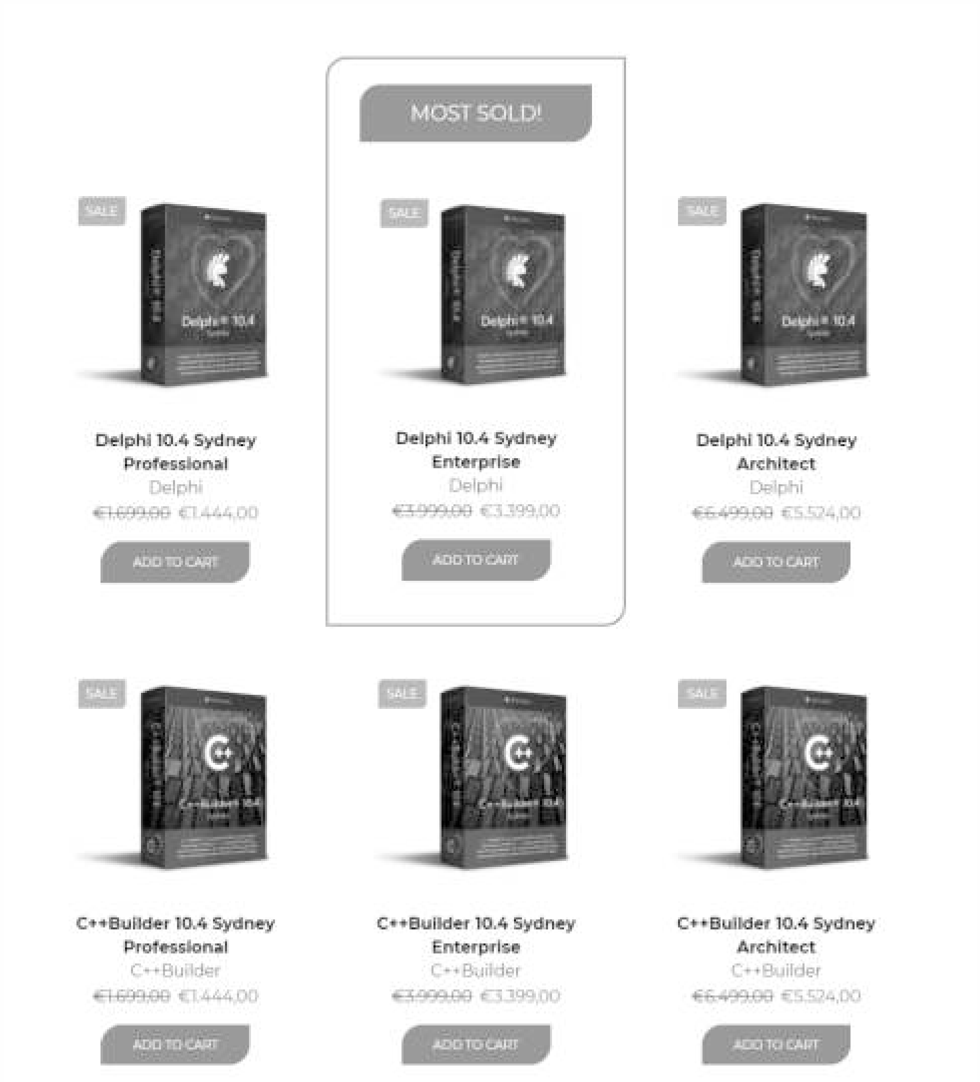
ex vat / inc. shipment

Promotions
Delphi & C++Builder are the best development tools on the market to design and develop modern, cross-platform native apps and services. It’s easier than ever to create stunning, high performing apps for Windows, macOS, iOS, Android and Linux Server (Linux Server is supported in Delphi Enterprise or higher), using the same native code base. Share visually designed UIs across multiple platforms that make use of native controls and platform behaviors, and leverage powerful and modern languages with enhancements that help you code faster.

SUPER DEALS AT BARNSTEN UNTIL January 31, 2021 *

Get 15% on RAD Studio, Delphi and C++Builder products
This can be bought directly in the webshop
PLUS

GET a FREE Web Pack of your choice when you buy a new license for Delphi/RAD Studio/C++Builder Enterprise or Architect. Choose between IntraWeb, TMS Web Core, or uniGUI and build amazing native applications. You will receive a link to redeem the Web Pack of your choice with the delivery of your license.
* These offers are not valid on Academic licenses and/or existing contracts



Delphi 10.4 Sydney Professional
Delphi
~~€1.699,00~~ €1.444,00

Delphi 10.4 Sydney Enterprise
Delphi
~~€3.999,00~~ €3.399,00

Delphi 10.4 Sydney Architect
Delphi
~~€6.499,00~~ €5.524,00

C++Builder 10.4 Sydney Professional
C++Builder
~~€1.699,00~~ €1.444,00

C++Builder 10.4 Sydney Enterprise
C++Builder
~~€3.999,00~~ €3.399,00

C++Builder 10.4 Sydney Architect
C++Builder
~~€6.499,00~~ €5.524,00

www.barnsten.com / info@barnsten.com
France: Téléphone +33 (0)9 72 19 28 87
Benelux: Telefoon +31 (0)2 35 42 22 27

RAD Studio 10.4 Sydney Professional
RAD
~~€2.999,00~~ €2.549,00

RAD Studio 10.4 Sydney Enterprise
RAD
~~€4.999,00~~ €4.249,00

RAD Studio 10.4 Sydney Architect
RAD
~~€6.999,00~~ €5.949,00

RAD Studio™ is the fastest way to design and develop modern, cross-platform native apps and services. It's easier than ever to create stunning, high performing apps for Windows, macOS, iOS, Android and Linux Server (Linux Server is supported in Delphi Enterprise or higher), using the same native code base. Share visually designed UIs across multiple platforms that make use of native controls and platform behaviors, and leverage powerful and modern languages with enhancements that help you code faster. Developers pick RAD Studio™ because it delivers 5x the speed for development and deployment across multiple desktop and mobile platforms.

Which promotion suits you the best?
Start today with the most powerful framework for Windows and native application development for Windows, macOS, Android, iOS and Linux.
You can choose for a license including 36 months subscription. You will get access to all new versions and technical support for three years.
You can choose option 2 if you want to bring your Delphi apps to the Web. You will receive your Delphi license incl. 1 year subscription and a free Web Component tool of your choice.

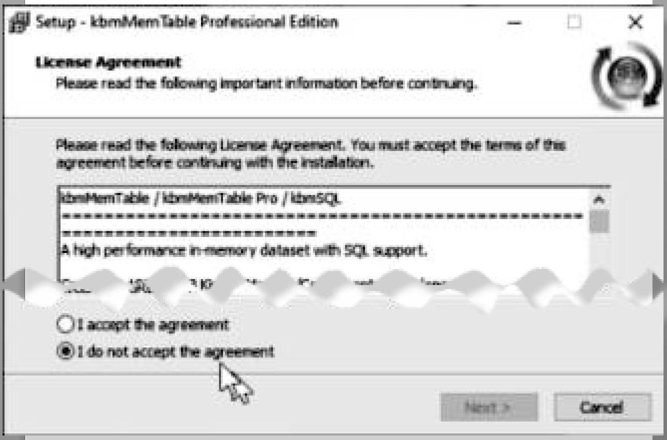
Chat with us or call if you have questions or need a personal advise.

Direct access to Windows 10 Store support
With the latest Embarcadero 10.3.3 Rio software you can transform existing and new Windows Desktop applications. The applications are suitable for the Microsoft Windows 10 Store, using the Desktop Bridge technology, also known as Centennial Bridge. In addition you will have to opportunity to sell to the entire world via the store.

www.barnsten.com / info@barnsten.com
France: Téléphone +33 (0)9 72 19 28 87
Benelux: Telefoon +31 (0)2 35 42 22 27

Compile Tool for kbmMemTable and kbmMW By Detlef Overbeek

Introduction:
I tried the newest extra from Kim Madsen: The compiletool.
It is a tool that you will get together with two programs: **`kbmMemTable`** and **`kbmMW`**.
The name is a bit misleading:
It actually is an installer that makes things that were rather complex before very easy.
In this overview I'll show where it is placed after installing the program of the MemTable or the Suite.
I am very pleased with this, not only because Component4Developers is an advertiser but because it makes it almost 100% easy to use it.
To convince you, these pages that show almost without text what to do and where you can find the installer of **"The Compile Tool".**


Setup - kbmMemTable Professional Edition

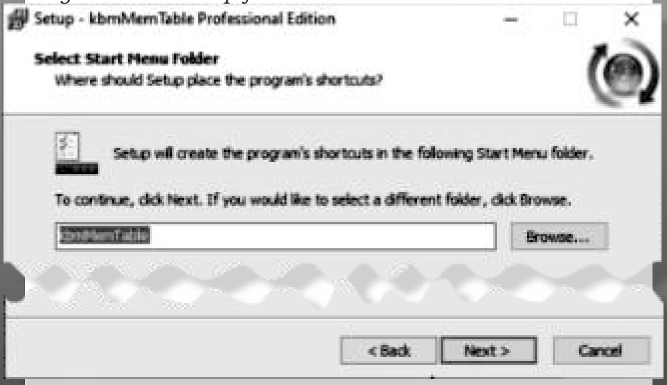
License Agreement
Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

kbmMemTable / kbmMemTable Pro / kbmSQL
==
A high performance in-memory dataset with SQL support.

○ I accept the agreement
● I do not accept the agreement

Figure 1: The setup for the memtable starts


Setup - kbmMemTable Professional Edition

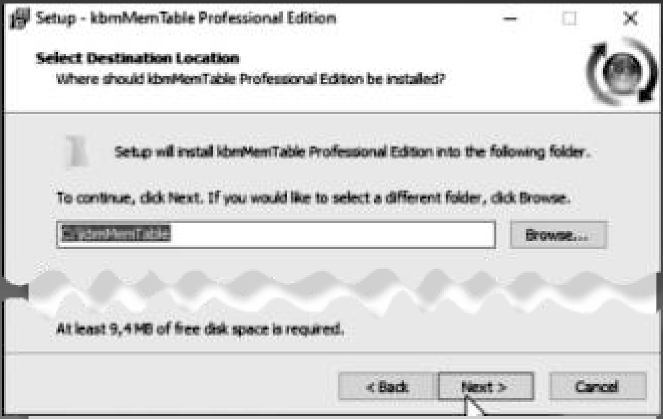
Select Start Menu Folder
Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

kbmMemTable

Figure 2: Program folder


Setup - kbmMemTable Professional Edition

Select Destination Location
Where should kbmMemTable Professional Edition be installed?

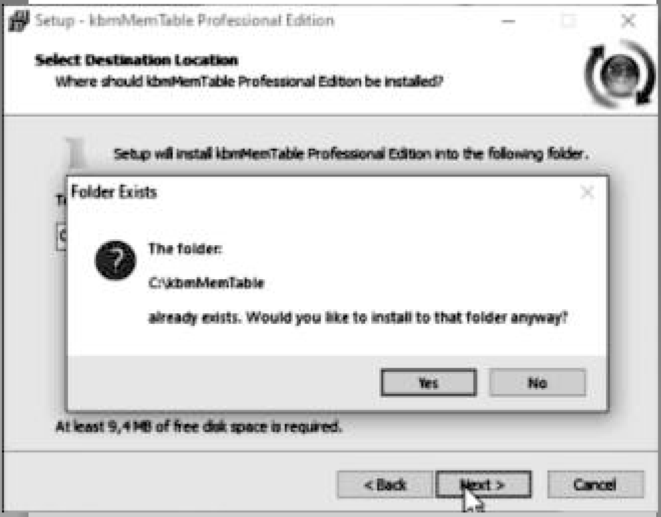
Setup will install kbmMemTable Professional Edition into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\kbmMemTable

At least 9,4 MB of free disk space is required.

Figure 3: Folder selection


Setup - kbmMemTable Professional Edition

Select Destination Location
Where should kbmMemTable Professional Edition be installed?

Setup will install kbmMemTable Professional Edition into the following folder.

Folder Exists

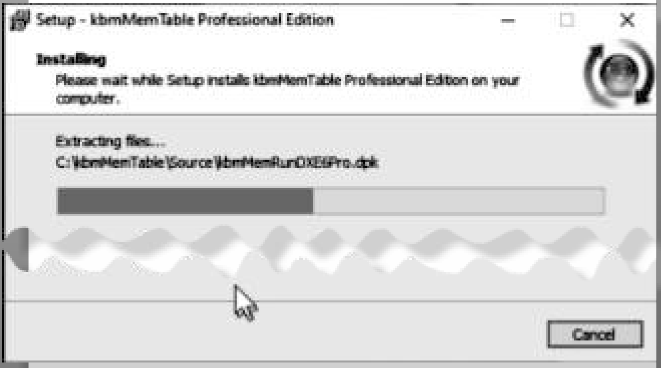
The folder:

C:\kbmMemTable

already exists. Would you like to install to that folder anyway?

At least 9,4 MB of free disk space is required.

Figure 4: If you had a previous installation make sure it is all removed and there is nothing of the installation is left

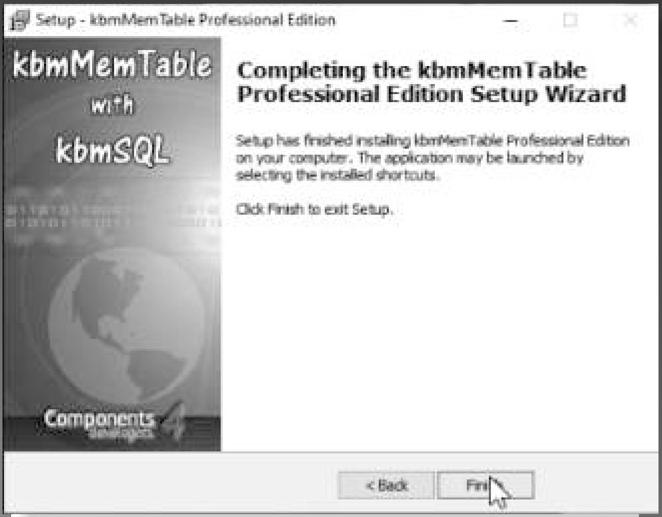

Setup - kbmMemTable Professional Edition

Installing
Please wait while Setup installs kbmMemTable Professional Edition on your computer.

Extracting files...
C:\kbmMemTable\Source\kbmMemRunDXE6Pro.dpk

Figure 5: The running installment

COMPONENTS DEVELOPERS 4

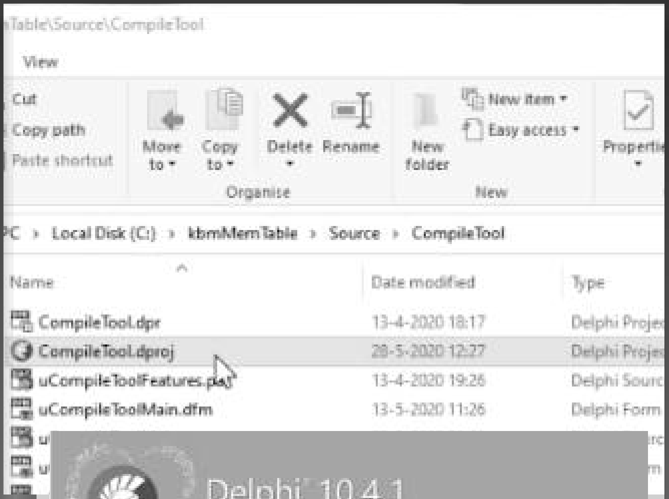

Setup - kbmMemTable Professional Edition

Completing the kbmMemTable Professional Edition Setup Wizard

Setup has finished installing kbmMemTable Professional Edition on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

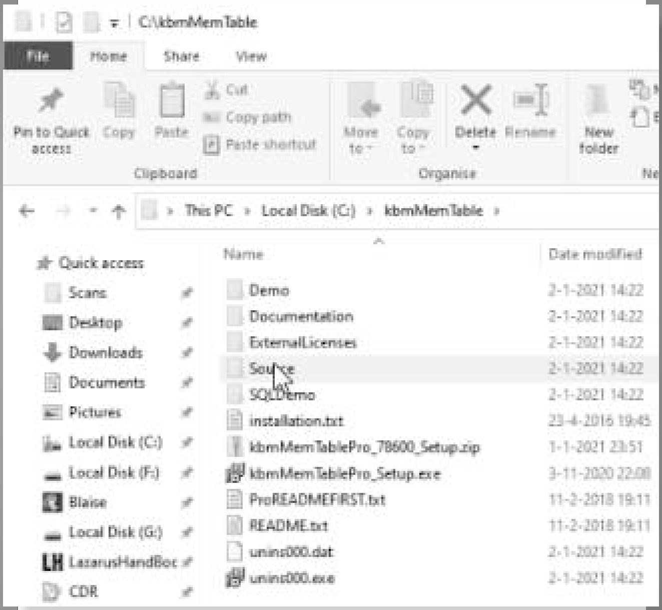
Figure 6: The installation is complete



PC › Local Disk (C:) › kbmMemTable › Source › CompileTool

Name	Date modified	Type
CompileTool.dpr	13-4-2020 18:17	Delphi Projec
CompileTool.dproj	28-5-2020 12:27	Delphi Projec
uCompileToolFeatures.pas	13-4-2020 19:26	Delphi Sourc
uCompileToolMain.dfm	13-5-2020 11:26	Delphi Form

Figure 9: It starts then Delphi


C:\kbmMemTable

This PC › Local Disk (C:) › kbmMemTable ›

Name	Date modified
Demo	2-1-2021 14:22
Documentation	2-1-2021 14:22
ExternalLicenses	2-1-2021 14:22
Source	2-1-2021 14:22
SQLDemo	2-1-2021 14:22
installation.txt	23-4-2016 19:45
kbmMemTablePro_78600_Setup.zip	1-1-2021 23:51
kbmMemTablePro_Setup.exe	3-11-2020 22:08
ProREADMEFIRST.txt	11-2-2018 19:11
README.txt	11-2-2018 19:11
unins000.dat	2-1-2021 14:22
unins000.exe	2-1-2021 14:22

Figure 7: In the source-dir you will find the Compiletool project

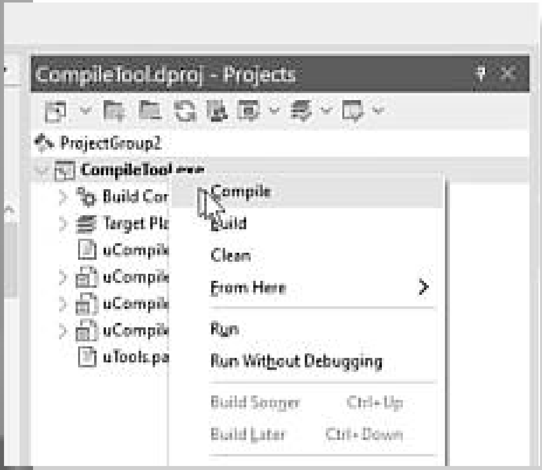
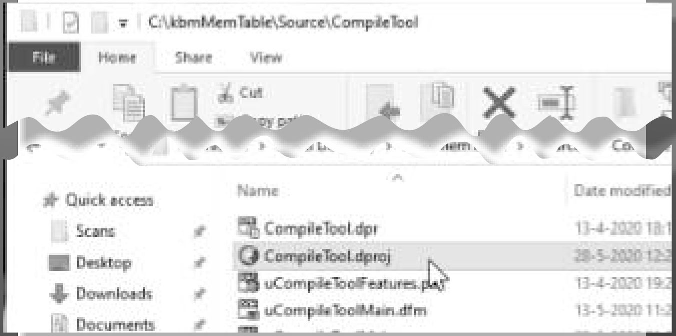
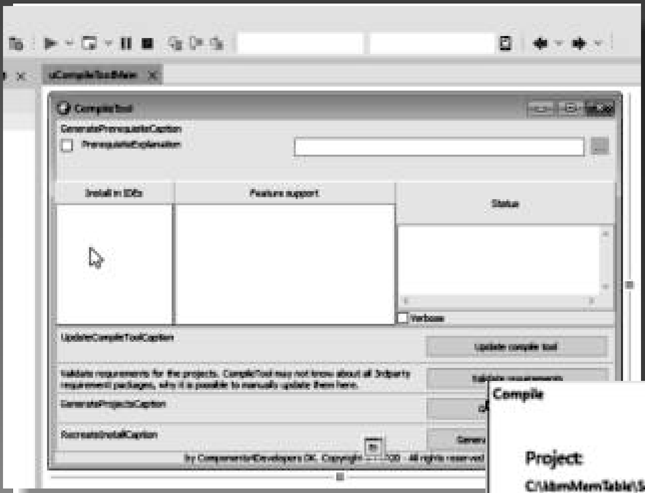

CompileTool.dproj - Projects

Figure 10: Compile the project (Right Column)


C:\kbmMemTable\Source\CompileTool

Name	Date modified
CompileTool.dpr	13-4-2020 18:1
CompileTool.dproj	28-5-2020 12:2
uCompileToolFeatures.pas	13-4-2020 19:2
uCompileToolMain.dfm	13-5-2020 11:2

Figure 8: Select the project and double-click



CompileTool

GeneratePrerequisiteCaption

PrerequisiteExplanation

Install in IDEs	Feature support	Status

Verbose

UpdateCompileToolCaption — Update compile tool

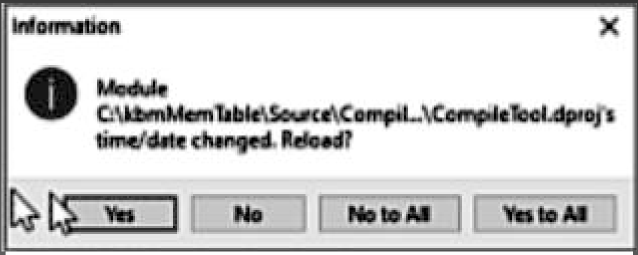
Validate requirements for the projects. CompileTool may not know about all 3rdparty requirement packages, why it is possible to manually update them here. — Validate requirements

GenerateProjectsCaption

RecreateInstallCaption

by Components4Developers DK. Copyright (c) 2020 - All rights reserved

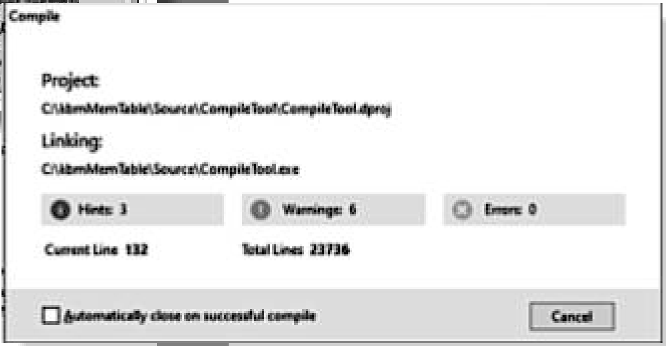
Figure 11: The form that will be compiled and generates the program.



Information

Module C:\kbmMemTable\Source\Compil...\CompileTool.dproj's time/date changed. Reload?

Figure 14: If you had compiled the program while Delphi was still running you can get this warning



Compile

Project:
C:\kbmMemTable\Source\CompileTool\CompileTool.dproj

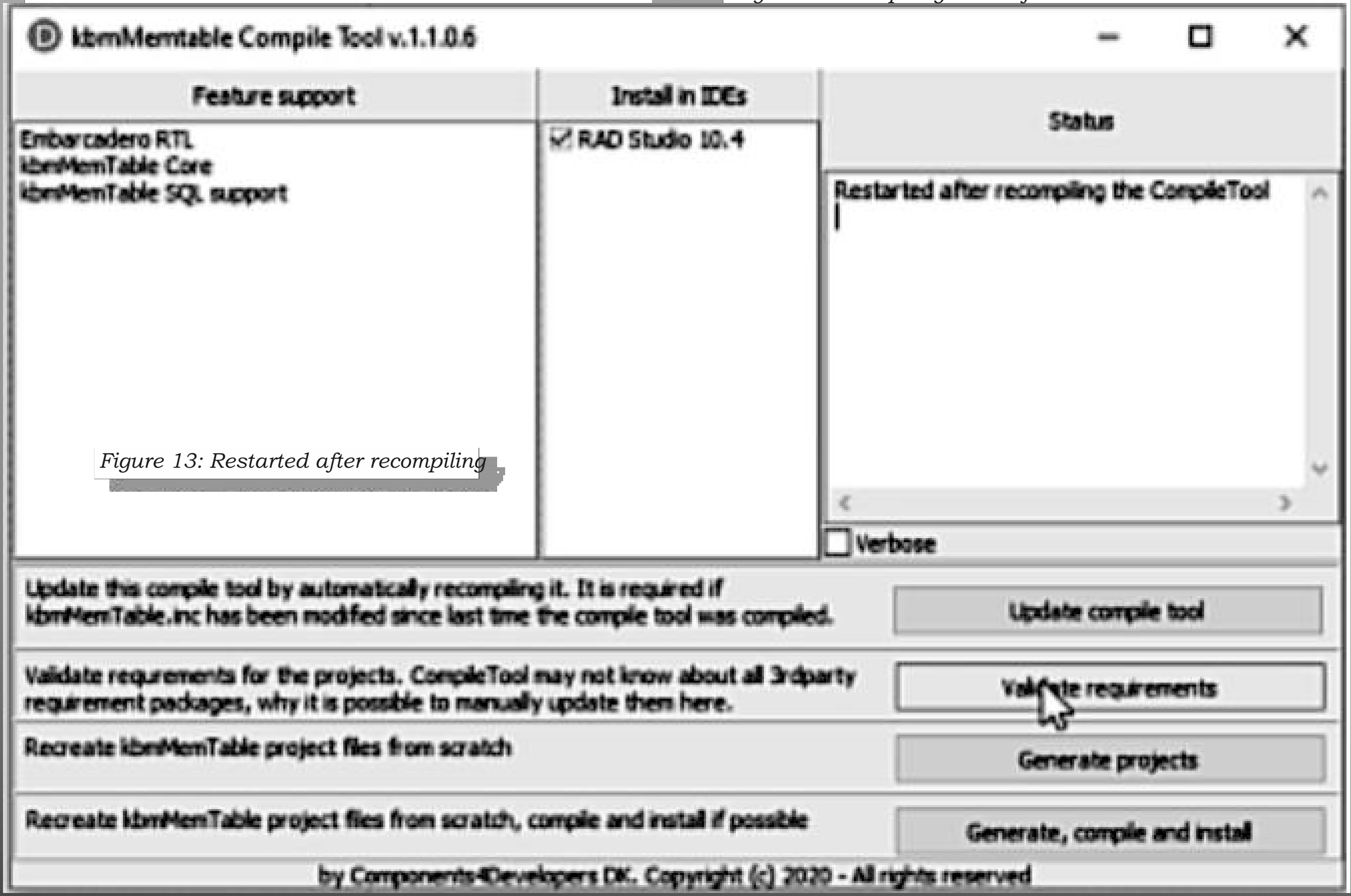
Linking:
C:\kbmMemTable\Source\CompileTool.exe

Hints: 3 Warnings: 6 Errors: 0

Current Line: 132 Total Lines: 23736

Automatically close on successful compile

Figure 12: Compiling the Project



kbmMemtable Compile Tool v.1.1.0.6

Feature support	Install in IDEs	Status
Embarcadero RTL kbmMemTable Core kbmMemTable SQL support	✓ RAD Studio 10.4	Restarted after recompiling the CompileTool

Verbose

Update this compile tool by automatically recompiling it. It is required if kbmMemTable.inc has been modified since last time the compile tool was compiled. — Update compile tool

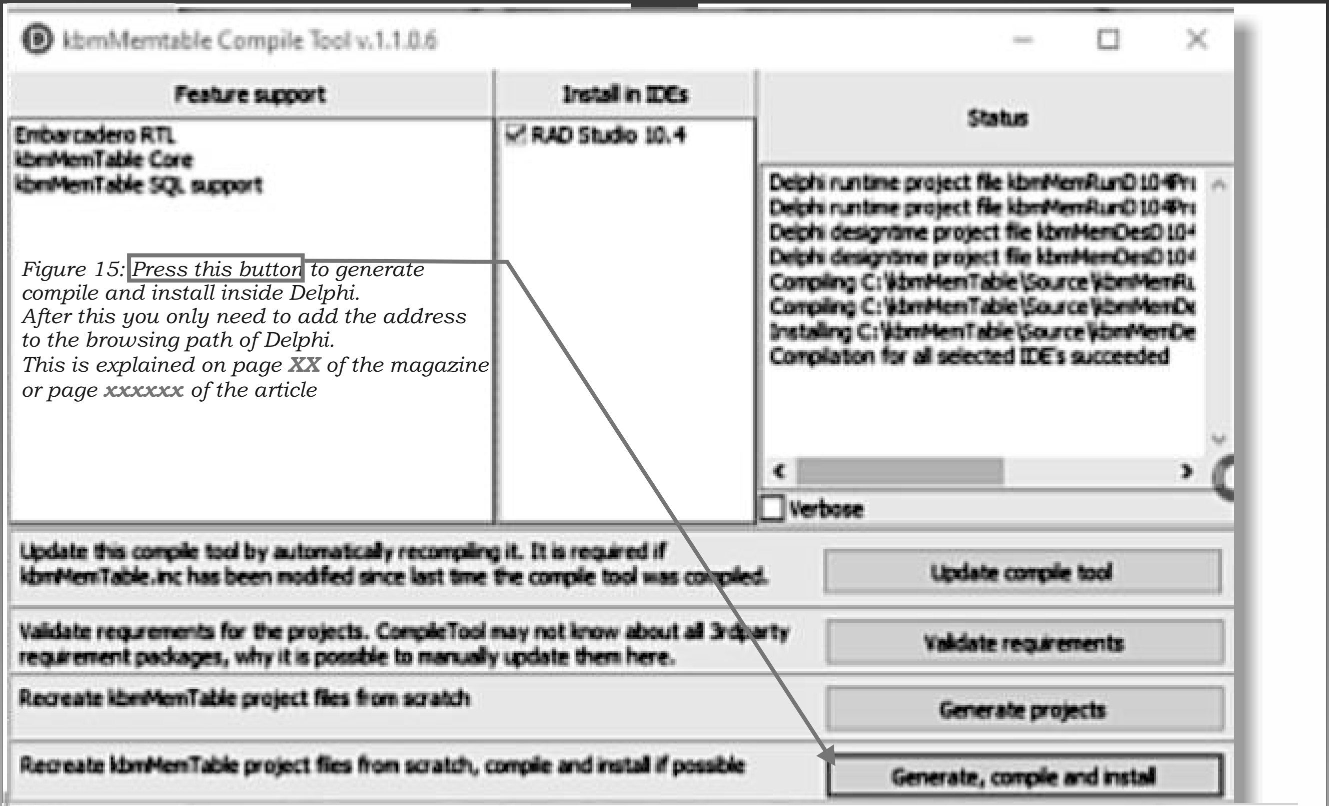
Validate requirements for the projects. CompileTool may not know about all 3rdparty requirement packages, why it is possible to manually update them here. — Validate requirements

Recreate kbmMemTable project files from scratch — Generate projects

Recreate kbmMemTable project files from scratch, compile and install if possible — Generate, compile and install

by Components4Developers DK. Copyright (c) 2020 - All rights reserved

Figure 13: Restarted after recompiling


kbmMemtable Compile Tool v.1.1.0.6

Feature support	Install in IDEs	Status
Embarcadero RTL kbmMemTable Core kbmMemTable SQL support	✓ RAD Studio 10.4	Delphi runtime project file kbmMemRunD104Pr Delphi runtime project file kbmMemRunD104Pr Delphi designtime project file kbmMemDesD104 Delphi designtime project file kbmMemDesD104 Compiling C:\kbmMemTable\Source\kbmMemRu Compiling C:\kbmMemTable\Source\kbmMemDe Installing C:\kbmMemTable\Source\kbmMemDe Compilation for all selected IDE's succeeded

Verbose

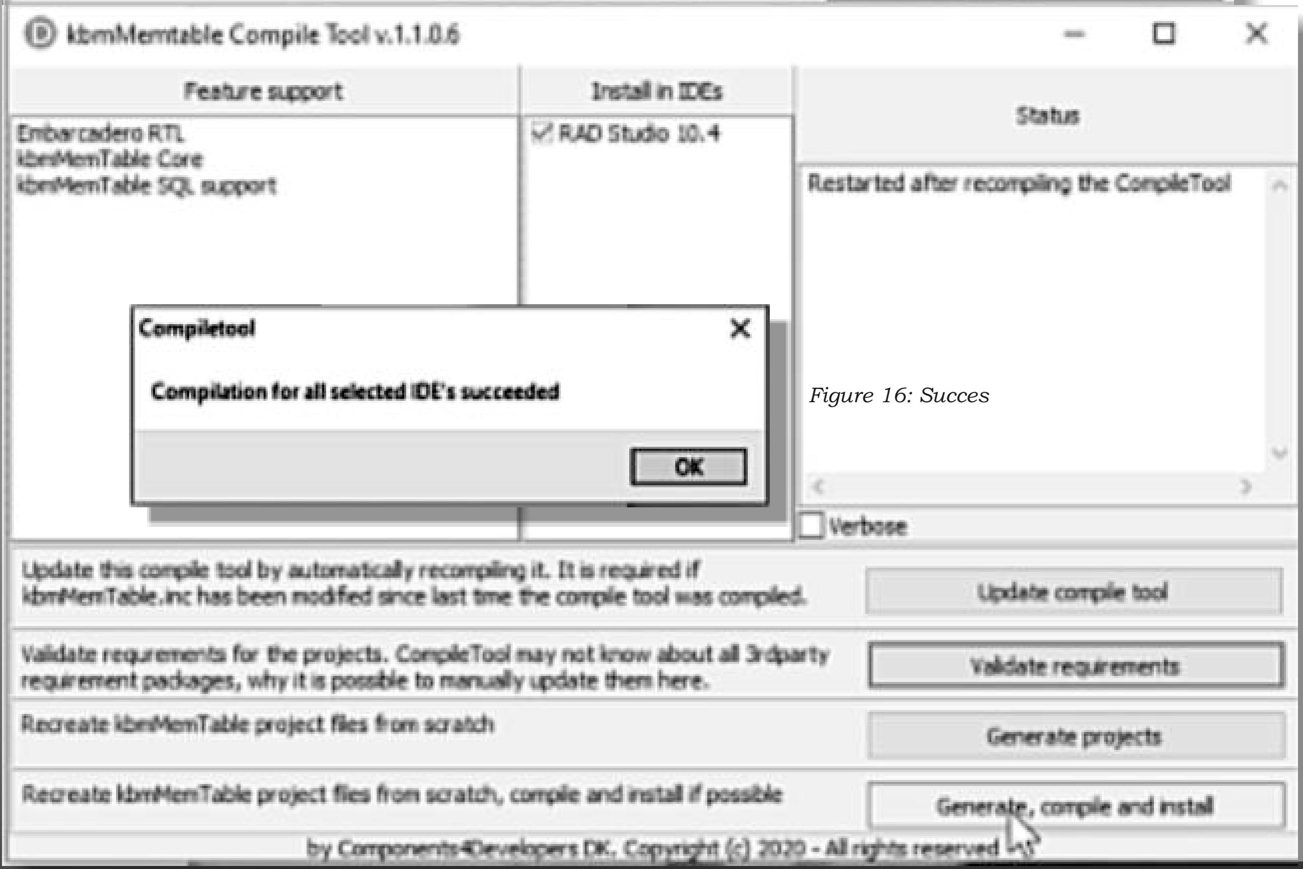
Update this compile tool by automatically recompiling it. It is required if kbmMemTable.inc has been modified since last time the compile tool was compiled. — Update compile tool

Validate requirements for the projects. CompileTool may not know about all 3rdparty requirement packages, why it is possible to manually update them here. — Validate requirements

Recreate kbmMemTable project files from scratch — Generate projects

Recreate kbmMemTable project files from scratch, compile and install if possible — Generate, compile and install

Figure 15: Press this button to generate compile and install inside Delphi.
After this you only need to add the address to the browsing path of Delphi.
*This is explained on page **XX** of the magazine or page **xxxxxx** of the article*


kbmMemtable Compile Tool v.1.1.0.6

Feature support	Install in IDEs	Status
Embarcadero RTL kbmMemTable Core kbmMemTable SQL support	✓ RAD Studio 10.4	Restarted after recompiling the CompileTool

Compiletool

Compilation for all selected IDE's succeeded

OK

Verbose

Update this compile tool by automatically recompiling it. It is required if kbmMemTable.inc has been modified since last time the compile tool was compiled. — Update compile tool

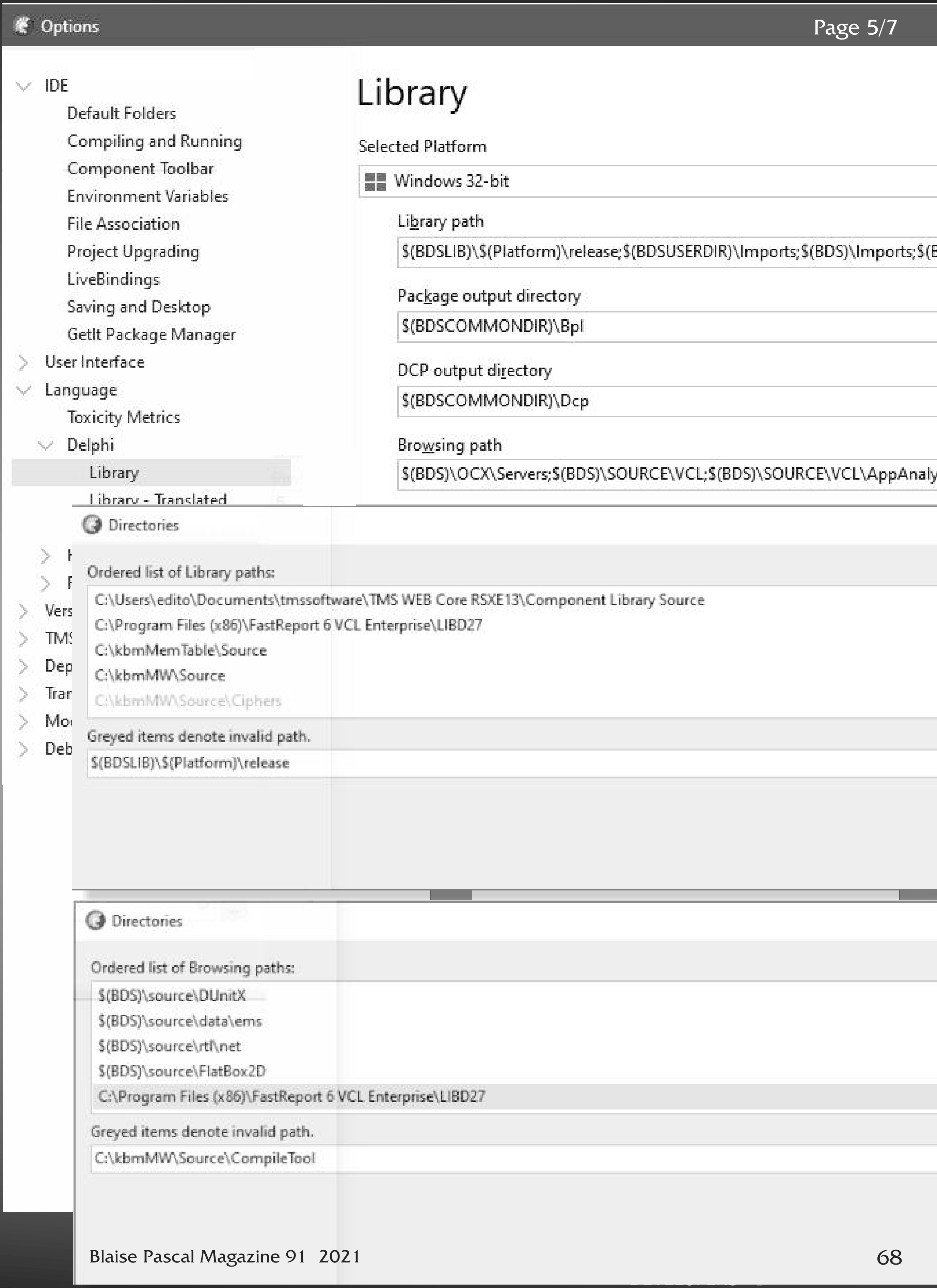
Validate requirements for the projects. CompileTool may not know about all 3rdparty requirement packages, why it is possible to manually update them here. — Validate requirements

Recreate kbmMemTable project files from scratch — Generate projects

Recreate kbmMemTable project files from scratch, compile and install if possible — Generate, compile and install

by Components4Developers DK. Copyright (c) 2020 - All rights reserved

Figure 16: Succes



Library

Selected Platform

Windows 32-bit

Library path

$(BDSLIB)\$(Platform)\release;$(BDSUSERDIR)\Imports;$(BDS)\Imports;$(B

Package output directory

$(BDSCOMMONDIR)\Bpl

DCP output directory

$(BDSCOMMONDIR)\Dcp

Browsing path

$(BDS)\OCX\Servers;$(BDS)\SOURCE\VCL;$(BDS)\SOURCE\VCL\AppAnaly

Directories

Ordered list of Library paths:

```
C:\Users\edito\Documents\tmssoftware\TMS WEB Core RSXE13\Component Library Source
C:\Program Files (x86)\FastReport 6 VCL Enterprise\LIBD27
C:\kbmMemTable\Source
C:\kbmMW\Source
C:\kbmMW\Source\Ciphers
```

Greyed items denote invalid path.

$(BDSLIB)\$(Platform)\release

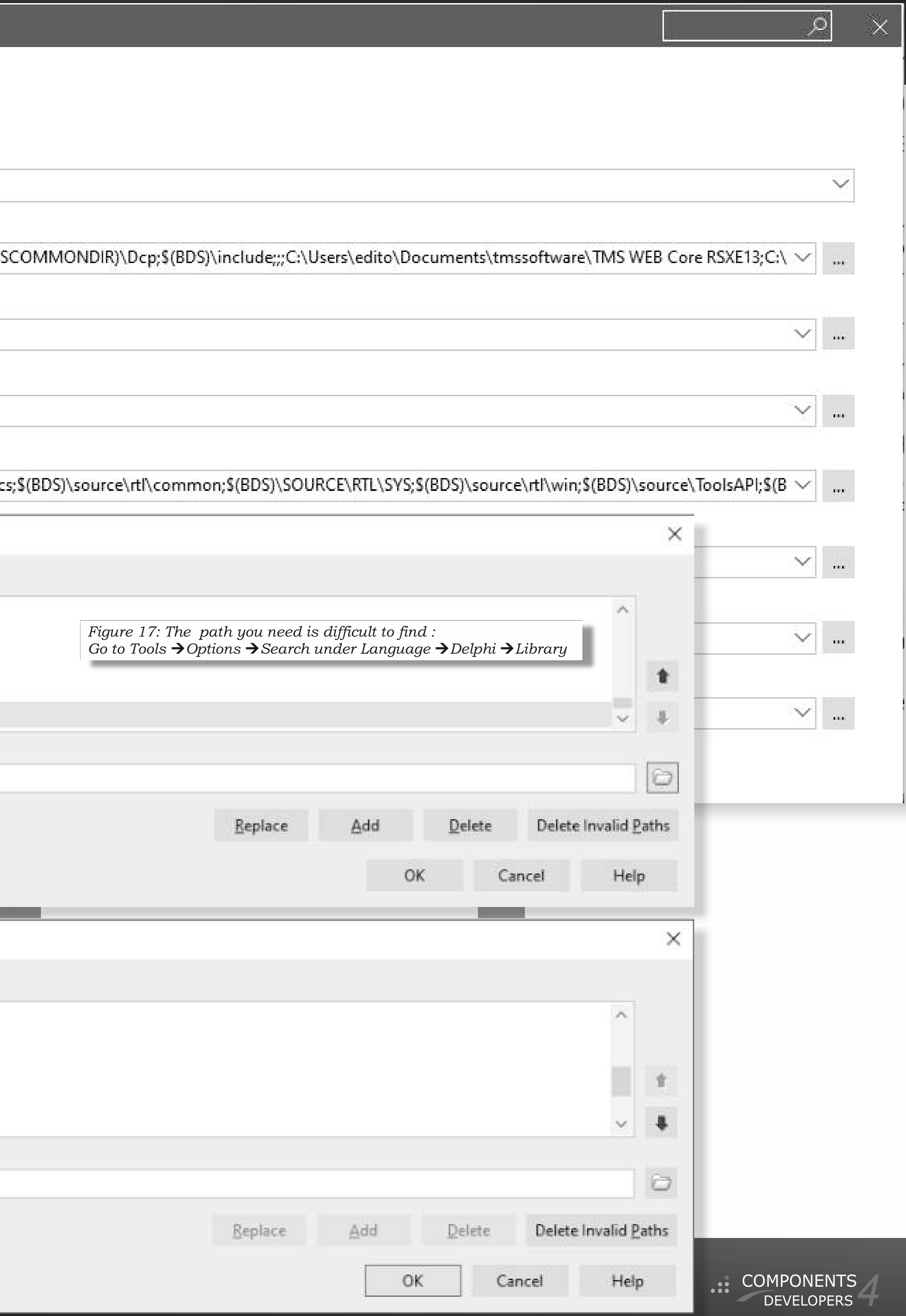
Directories

Ordered list of Browsing paths:

```
$(BDS)\source\DUnitX
$(BDS)\source\data\ems
$(BDS)\source\rtl\net
$(BDS)\source\FlatBox2D
C:\Program Files (x86)\FastReport 6 VCL Enterprise\LIBD27
```

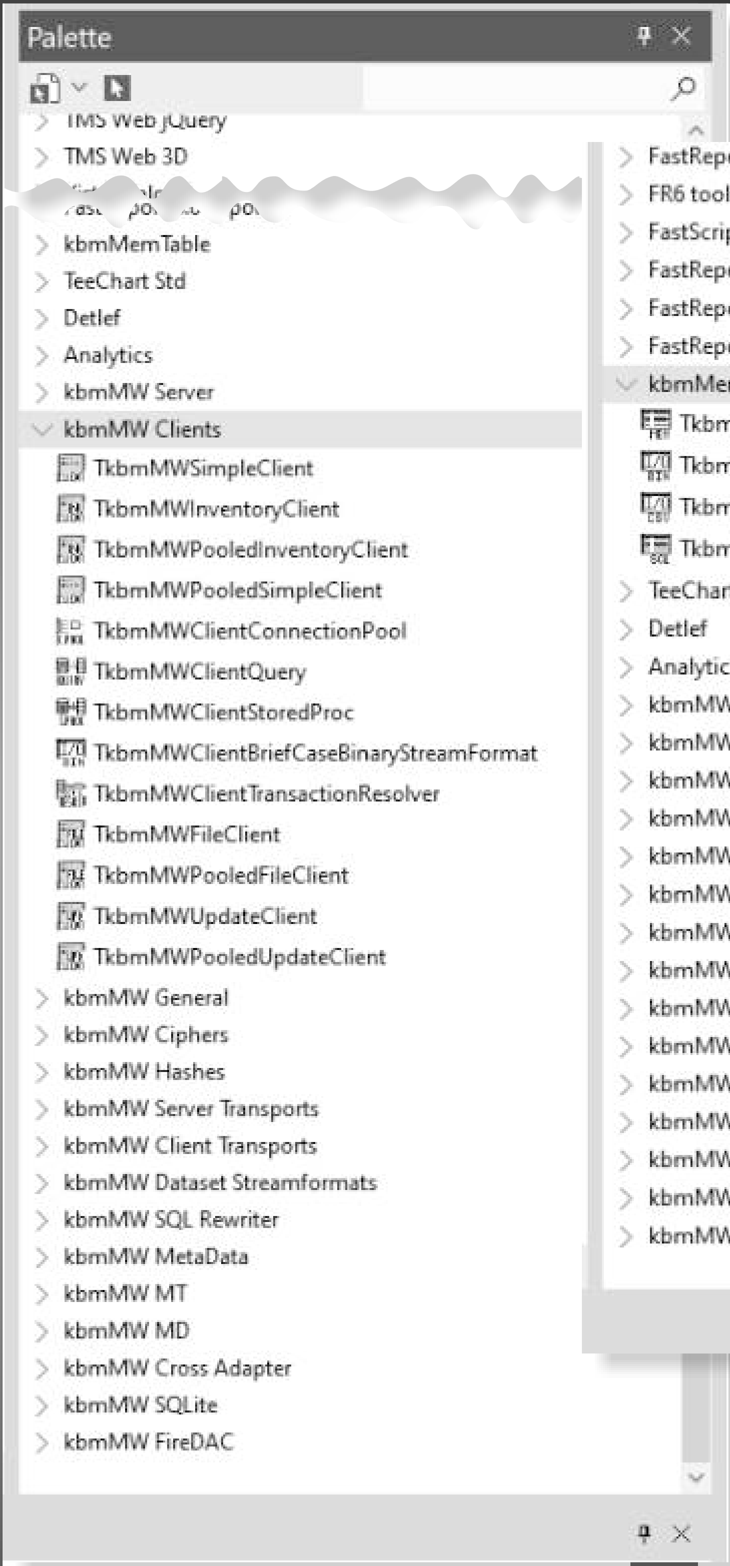
Greyed items denote invalid path.

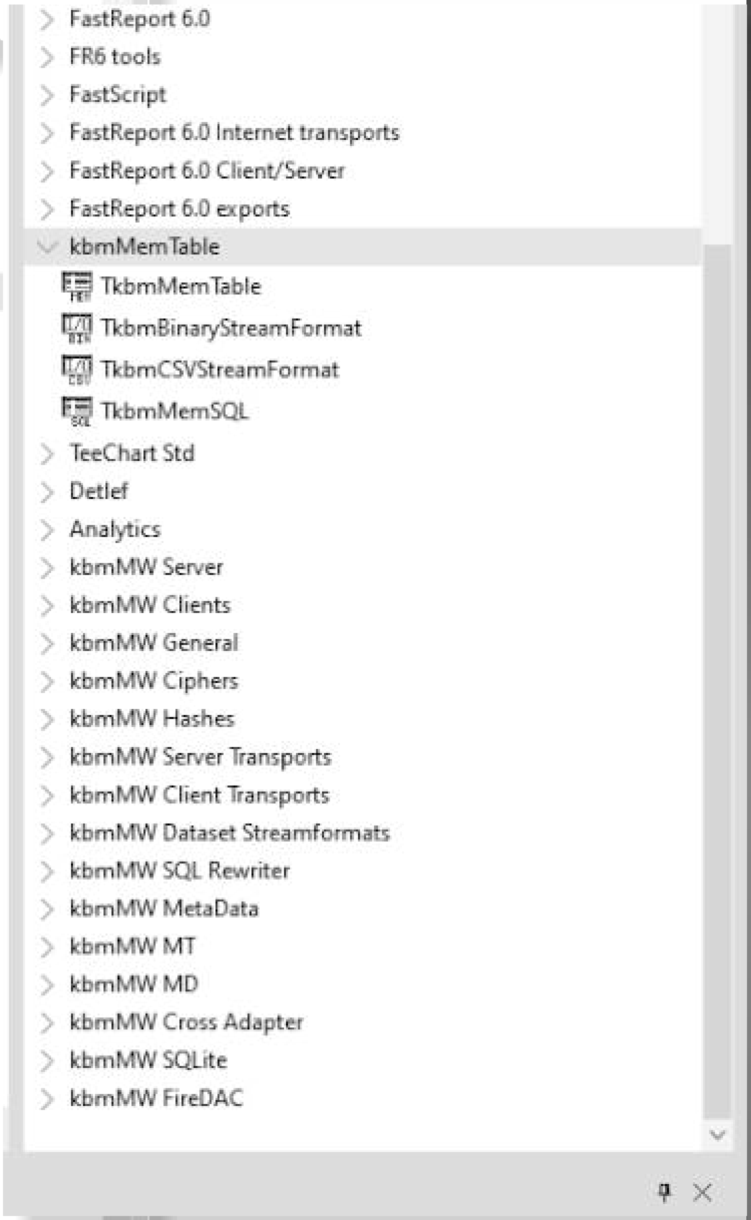
C:\kbmMW\Source\CompileTool


SCOMMONDIR)\Dcp;$(BDS)\include;;;C:\Users\edito\Documents\tmssoftware\TMS WEB Core RSXE13;C:\

cs;$(BDS)\source\rtl\common;$(BDS)\SOURCE\RTL\SYS;$(BDS)\source\rtl\win;$(BDS)\source\ToolsAPI;$(B

Figure 17: The path you need is difficult to find :
Go to Tools →Options →Search under Language →Delphi →Library





Here you see all the components that where installed with the help of the compile tool.
How easy can it be?
The old way of installing - a bit more complex - is still possible

Are Bumblebees picky?

When it comes to feeding on pollen, honeybees and bumblebees are generalists. They like a buffet of choices – except when it comes to pollen from flowers of the genus Cucurbita, including squash and pumpkin, which they avoid.

The Cornell study: “**Pollen Defenses Negatively Impact Foraging and Fitness in a Generalist Bee,**” published Feb. 20 in the journal **Nature Scientific Reports**, found that squash and pumpkin pollen have physical, nutritional and chemical defense qualities that are harmful to bumblebees. When bumblebees are fed cucurbit pollen, it causes all kinds of problems, Adults have damaged and distorted digestive tracts and colonies fed cucurbit pollen failed to rear any offspring.
Bumblebees do visit pumpkin and squash flowers for the nectar, and though they don’t collect the pollen, some might inadvertently get on their legs.They were actually seen in the field using their legs to groom it off their bodies and then wipe it on a leaf. Not only are they not collecting it, they actually hate it. The [cucurbit] system is really interesting because we have specialists and generalist bees feeding on the same resource. The results suggest that deterring bumblebees from collecting and eating pollen may provide an evolutionary benefit to cucurbit plants.

Bees that are really effective at collecting and eating certain types of pollen may be actually functioning more like herbivores and pollen thieves than actual pollinators. At the same time, bees that visit plants for nectar but don’t collect pollen may be good pollinators, as stray pollen on their bodies may end up pollinating the next flower.

What this tells us is that some plant pollen may be chemically or mechanically protected from generalist bees which, oddly enough, can benefit the plants in terms of pollination. In the study, Brochu and colleagues created diets that represented different defenses to test which cucurbit pollen characteristics deterred bumblebees.
One diet of wildflower pollen collected by honeybees served as a control. A second consisted of unadulterated cucurbit pollen, which is nutritionally poor food for bumblebees, has large and spiny grains, and contains natural chemicals. In a third treatment, the team extracted the chemicals from the cucurbit pollen and added them to the control diet of nutritionally rich wildflower pollen.

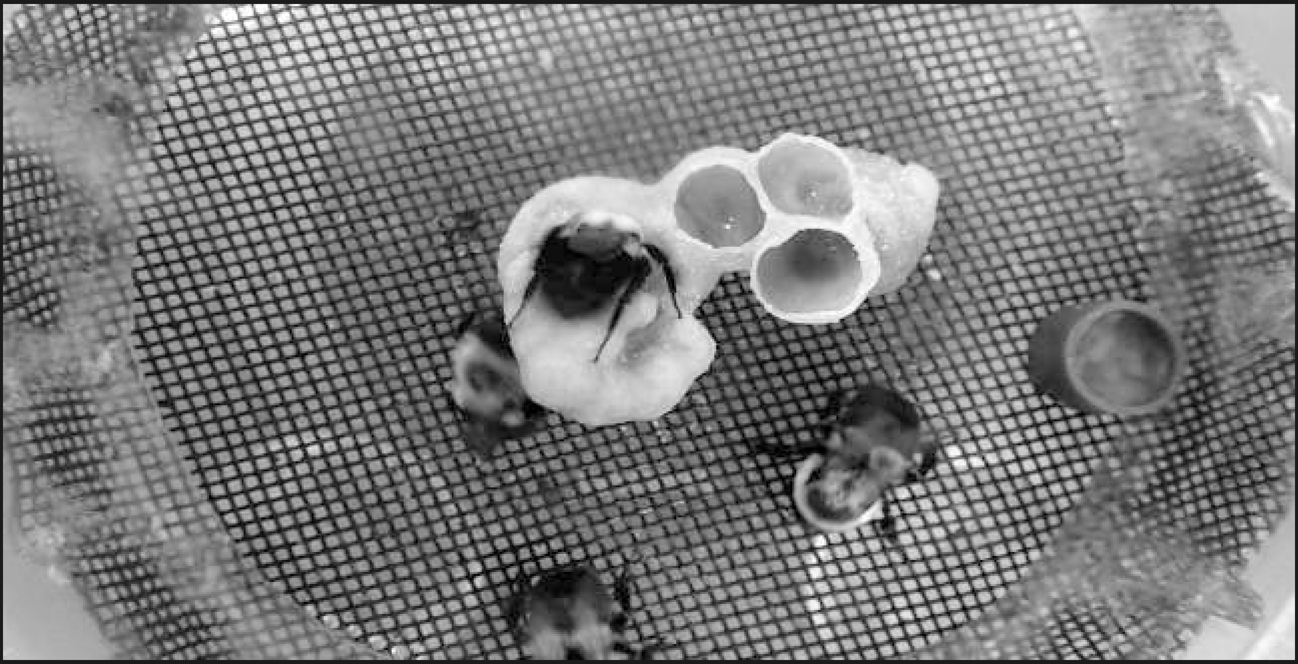
Microcolonies of five bees were each fed a separate treatment. The bees fed the wildflower pollen thrived, as expected. Under a natural cucurbit diet, the cumulative effect of the pollen’s physical defenses, poor nutritional content and chemicals led to bees ejecting their offspring from their brood cells and killing them. Bumblebees do this when stressed, possibly because they can’t take care of the larvae, Brochu said.

With the crushed pollen diet, where the pollen’s physical defenses were removed, eggs and larvae failed to mature. Over the course of the 50 days of the experiment, in both the crushed and natural cucurbit treatment, no offspring made it to adulthood.

In the crushed treatment, the adults also died at a higher rate, possibly due to a release of additional toxic chemicals. And with the chemical wildflower treatment, larvae made it to adulthood most of the time and the bees ate more, possibly to compensate for something in the chemicals. Their abdomens became hard and dark, a process called melanization, which indicated trauma to the guts.

We tend to think that all pollen resources are great for all bees, but I don’t think that’s true.

For the sake of bumblebees pumpkin and squash growers may think twice about bringing commercial bumblebees into their fields and may provide wildflower strips as alternative food sources. Bumblebees avoided gathering cucurbit pollen.



KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.13 RELEASED!

- **RAD Studio XE5 to 10.4.1 Sydney supported**
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OSX client and server support
- Native high performance 100% developer defined application server
- Full support for centralized and distributed load balancing and failover
- Advanced ORM/OPF support including support of existing databases
- Advanced logging support
- Advanced configuration framework
- Advanced scheduling support for easy access to multithread programming
- Advanced smart service and clients for very easy publication of functionality
- High quality random functions.
- High quality pronouncable password generators.
- High performance LZ4 and Jpeg compression
- Complete object notation framework including full support for YAML, BSON, Messagepack, JSON 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**

♦ **Improvement to ORM including support for rewriting JOINS.**

♦ **Improvements to SmartBind**

♦ **Improvements to SmartEvent**

♦ **Improvements to LINQ**

♦ **Support of XML mixed text documents**

♦ **New TkbmMWXMLReformatter which can analyze and reformat even damaged XML**

♦ **New support for USB/HID devices including MagTek barcode reader class (Windows/Ent)**

♦ **More features 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

DX

EESB, SOA,MoM, EAI TOOLS FOR INTELLIGENT SOLUTIONS. kbmMW IS THE PREMIERE N-TIER PRODUCT FOR DELPHI / C++BUILDER